

CORNELIU HUTANU

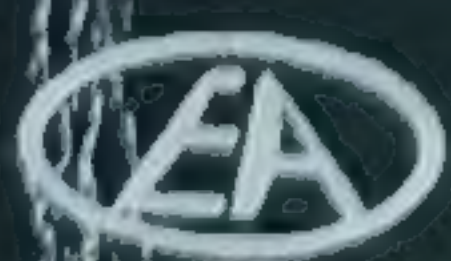
MIHAI POSTOLACHE

SISTEME CU MICROPROCESOARE

ÎN CONDUCEREA AUTOMATĂ A PROCESELOR

Volumul I

Ediția a 2-a (revăzută)



EDITURA
ACADEMICA

IASI - 2001

Butea

05.2002

CORNELIU HUȚANU

MIHAI POSTOLACHE

SISTEME CU MICROPROCESOARE

ÎN CONDUCEREA AUTOMATĂ A PROCESELOR

Volumul 1

Ediția a 2-a (revăzută)

IAȘI - 2001

CORNELIU HUȚANU

MIHAI POSTOLACHE

SISTEME CU MICROPROCESOARE ÎN CONDUCEREA AUTOMATĂ A PROCESELOR

Volumul 1

Ediția a 2-a (revăzută)



EDITURA ACADEMICA
IAȘI - 2001

Prefață la Ediția a II-a

Apariția microprocesoarelor, ca rezultat al rafinării tehnologiilor electronice de integrare pe corp solid, a impulsionat practic toate domeniile societății umane. Îmbinând avantajele tehnicilor numerice de prelucrare a informației cu flexibilitatea dată de programabilitate, microprocesorul a determinat mutații majore în abordarea conceptuală a diferitelor realizări tehnice. În fond, un microprocesor are elementele esențiale ale unei unități centrale de procesare (UCP), realizată obișnuit sub forma unui monocip. Natural, impactul imediat s-a resimțit asupra evoluției (micro)calculatoarelor numerice. Folosirea microprocesoarelor la realizarea acestora a permis reducerea dramatică a prețurilor, simultan cu creșterea fiabilității.

Deși utilizarea calculatoarelor numerice universale în conducerea automată a proceselor tehnologice are o relativ lungă istorie (primele abordări se situează în deceniul șapte), folosirea microprocesoarelor a permis cu adevărat o abordare eficientă. În acest mod devine posibilă aplicarea conceptului modern de conducere distribuită, care practic nu este realizabil economic cu calculatoare clasice.

Evoluția tehnologiilor de integrare a permis nu numai creșterea puterii de calcul a microprocesoarelor și realizarea unor UCP monocip deosebit de performante, dar și orientarea spre microsisteme integrate pe un singur cip, organizate avantajos pentru aplicații de control.

Astfel, tehnica microprocesoarelor influențează direct și avantajos atât abordarea teoretică a conducerii automate (conducerea distribuită în timp real, algoritmi distribuiți de conducere ș.a.), precum și eficiența implementării, cu incidență asupra materializării conceptului de automatizare de cost redus ("Low Cost Automation"). Se poate spune că microprocesoarele constituie elementele de bază ale introducerii cu adevărat a informaticii în automatizarea proceselor.

Adresându-se în principal studenților secției de Automatică și Informatică Industrială de la Facultatea de Automatică și Calculatoare din Iași, lucrarea este astfel

concepută încât să fie utilă unei categorii mai largi de cititori interesați de domeniul microprocesoarelor. Aceasta jalonează principalele elemente privind resursele fizice și logice ale sistemelor bazate pe microprocesoare, prin prisma cerințelor specifice conducerii automate a proceselor și instalațiilor tehnologice.

Concepută în două volume, lucrarea se dorește a fi o dezvoltare graduală a principalelor tehnici utilizate în structurarea hardware și software a sistemelor cu microprocesoare. În volumul de față (primul) sunt prezentate aspectele de bază ale tehnicii microprocesoarelor, cu exemplificări din categoria sistemelor bazate pe microprocesoare de 8 biți. Evoluția deosebită în acest domeniu nu permite o abordare exhaustivă a problemei, motiv pentru care s-a insistat numai asupra unor microprocesoare de referință (Intel - 8080/8085) sau performante (Zilog - Z80). Sunt dezvoltate principalele aspecte legate de utilizarea microprocesoarelor în conducerea automată, cum ar fi: tratarea cererilor multiple de întreruperi, introducerea timpului în prelucrarea informației și accesul direct la memorie. Un spațiu corespunzător se acordă conectării perifericelor la sistem, precum și comunicației seriale în sistemele cu microprocesoare. Expunerea este însoțită de numeroase exemple, alese astfel ca să permită o mai bună înțelegere a problematicii tratate. S-a acordat mai multă atenție aplicațiilor bazate pe 8085 și Z80, ca reprezentanți actuali ai microprocesoarelor de uz general de opt biți.

În cea de-a II-a ediție, pe lângă o corectare și revizuire a materialului din prima ediție, s-a înlocuit conținutul capitolului 6, care trata microcontrolerele din familia MCS-48, cu familia MCS-51, mai actuală și cu o mai largă răspândire în conducerea automată. Deși microcontrolerul 8048 a fost special conceput pentru aplicații locale de control, succesorul 8051/8052 aduce o concepție nouă și un plus de flexibilitate, ceea ce se regăsește în "longevitatea" sa: după peste 20 de ani de la apariție este încă unul dintre cele mai utilizate și apreciate microcontrolere din această gamă. Cele menționate sunt justificate și prin faptul că mulți alți producători au adoptat "nucleul" familiei MCS-51 și au produs noi microcontrolere deosebit de apreciate, cum ar fi: 80C522 al firmei Philips, SAB80535 al firmei Siemens, sau produsele firmelor Dallas Semiconductor Corp. și Atmel, având resurse "on chip" care le fac deosebit de atractive pentru implementarea aplicațiilor de control local sau distribuit.

Cuprins

Prefață la Edția a II-a	V
Cuprins	VII
Partea I - Microprocesoare în conducerea automată a proceselor	1
Cap.1. Structura sistemelor cu microprocesoare	3
1.1. Componentele principale ale unui sistem cu microprocesor	4
1.1.1. Microprocesorul	5
1.1.1.1. Stare mașină. Ciclu mașină. Ciclu instrucțiune	7
1.1.1.2. Magistralele unui microprocesor	8
1.1.1.3. Regimuri de funcționare ale microprocesoarelor	9
1.1.1.4. Programarea microprocesoarelor	11
1.1.1.5. Evoluția microprocesoarelor. Tipuri principale	16
1.1.2. Memoria sistemelor cu microprocesoare	20
1.1.2.1. Transferul de informație între memorie și microprocesor	21
1.1.2.2. Conectarea memoriilor statice la magistrala sistemului	22
1.1.2.3. Conectarea memoriilor RAM dinamice (DRAM)	26
1.1.2.4. Memoria stivă	30
1.1.3. Porturi de intrare-ieșire	31
1.1.3.1. Transferul de informație dintre UCP și porturile I/E	31
1.1.3.2. Conectarea porturilor I/E la magistrala sistemului	32
Cap.2. Interfațarea sistemelor cu microprocesoare	36
2.1. Interfațarea paralelă	36
2.2. Interfațarea serială	38
2.3. Tehnici de intrare-ieșire	40
2.3.1. Transferul programat intrare-ieșire	40
2.3.2. Transferul intrare-ieșire cu utilizarea întreruperilor	41
2.3.3. Transferul intrare-ieșire prin acces direct la memorie	42
2.4. Tehnici de introducere a timpului ca variabilă în prelucrarea informației	43
Cap.3. Comunicația serială în sistemele cu microprocesoare	45
3.1. Elemente și concepte de bază	45
3.1.1. Protocoale de comunicație	45
3.1.2. Modelul general al comunicației seriale punct la punct	47
3.1.2.1. Parametrii comunicațiilor seriale	48
3.1.3. Modelul general al comunicației seriale multipunct	49
3.1.3.1. Comunicația serială multipunct de tip master-slave	50
3.1.4. Codificarea informației în comunicațiile seriale	53
3.1.4.1. Detecția și corecția erorilor	53
3.2. Adaptoare de comunicație serială	55
3.2.1. Standardul RS-232	56
3.2.1.1. Interfața TTL - RS-232	58
3.2.1.2. Interconectarea echipamentelor compatibile RS-232	58

3.2.1.3. Protocolul XON/XOFF	59
3.2.2. Standardele RS-423A și 422A	59
3.2.3. Standardul RS-485	60
3.3. Tipuri de comunicație serială	62
3.3.1. Comunicația serială asincronă (START-STOP)	62
3.3.2. Comunicația serială sincronă	64
3.3.3. Protocele de date	64
3.3.3.1. Protocele de comunicație orientate pe caracter (BCP)	65
3.3.3.2. Protocele de comunicație orientate pe bit (BOP)	66
Partea a II-a - Sisteme cu microprocesoare de 8 biți	68
Cap.4. Sisteme cu microprocesoare Intel de 8 biți	69
4.1. Sisteme bazate pe microprocesorul Intel 8080A	69
4.1.1. Microprocesorul 8080A. Caracteristici generale	69
4.1.2. Secvențierea operațiilor interne	73
4.1.3. Tipuri de cicluri mașină. Octetul de stare	76
4.1.4. Unitatea centrală de procesare cu 8080A	77
4.1.4.1. Circuitul pentru generarea semnalelor de tact și de comandă	78
4.1.4.2. Circuitul pentru generarea magistralei de control	79
4.1.4.3. Structura unității centrale și a unui sistem cu 8080A	80
4.2. Sisteme cu microprocesorul Intel 8085A	82
4.2.1. Secvențierea operațiilor interne la microprocesorul 8085A	84
4.2.2. Structura sistemelor cu 8085A realizate cu dispozitive standard	87
4.2.3. Structura sistemelor cu 8085A folosind circuite dedicate	89
4.3. Programarea microprocesoarelor 8080A și 8085A	93
4.3.1. Formatul instrucțiunilor	93
4.3.2. Moduri de adresare	93
4.3.3. Setul de instrucțiuni	96
4.3.4. Programarea în limbaj de asamblare	101
4.3.4.1. Limbajul de asamblare Macro-80	101
4.3.4.2. Tehnici de programare	104
4.4. Sisteme pentru tratarea cererilor multiple de întrerupere	110
4.4.1. Sisteme de tratare a întreruperilor bazate pe tehnica de interogare	110
4.4.2. Sisteme de întreruperi vectorizate	112
4.4.2.1. Controlerele de întreruperi vectorizate 8259/8259A	112
4.4.2.2. Organizarea SINT cu PIC 8259/8259A	118
4.5. Introducerea timpului în prelucrarea informației	124
4.5.1. Circuitul de timp 8253	124
4.5.1.1. Programarea dispozitivului 8253	124
4.5.1.2. Modurile de operare ale PIT 8253	125
4.5.2. Organizarea unui SIT cu dispozitive 8253	126
4.6. Accesul direct la memorie în sistemele cu microprocesoare Intel de 8 biți	129
4.6.1. Controlerul pentru accesul direct la memorie 8257	133
4.6.1.1. Realizarea transferului DMA cu 8257	134
4.6.1.2. Programarea circuitului 8257	138
4.6.1.3. Organizarea accesului direct la memorie cu circuitul 8257	140
4.6.2. Controlerul 8237A	142
4.6.2.1. Moduri de transfer la 8237A	144
	146

Sisteme cu microprocesoare în conducerea automată a proceselor

4.6.2.2. Tipuri de transfer DMA la 8237A	147
4.6.2.3. Efectuarea transferului DMA cu 8237A	147
4.6.2.4. Registrele interne ale circuitului 8237A	148
4.6.2.5. Organizarea accesului direct la memorie folosind 8237A	151
4.7. Interfațarea cu periferice în sisteme cu microprocesoare Intel	152
4.7.1. Interfața paralelă 8255/8255A	152
4.7.1.1. Moduri de operare	153
4.7.1.2. Programarea dispozitivului 8255A	156
4.7.2. Utilizarea circuitului 8255A	157
4.8. Comunicația serială în sisteme cu microprocesoare Intel.....	160
4.8.1. Interfața serială 8251/8251A	160
4.8.1.1. Funcționarea circuitului 8251A în modul de lucru asincron	161
4.8.1.2. Modul de lucru sincron	162
4.8.1.3. Programarea circuitului 8251A	164
4.8.2. Utilizarea circuitului USART 8251A	166
Cap.5. Sisteme cu microprocesorul Z80	171
5.1. Microprocesorul Z80	171
5.1.1. Conexiunile externe ale microprocesorului Z80	174
5.1.2. Tratarea întreruperilor la microprocesorul Z80	176
5.1.2.1. Tratarea întreruperilor nemascabile	177
5.1.2.2. Tratarea întreruperilor mascabile	178
5.1.3. Secvențierea operațiilor interne la microprocesorul Z80	182
5.1.3.1. Ciclul de extragere a codului operației	182
5.1.3.2. Ciclurile de citire/înscrisoare a memoriei	183
5.1.3.3. Ciclurile de intrare/ieșire	184
5.1.3.4. Ciclul de cerere/confirmare a întreruperii mascabile	185
5.1.3.5. Ciclul de cerere/răspuns la o întrerupere nemascabilă	187
5.1.3.6. Ciclul de cerere/confirmare cedare de magistrale	188
5.1.3.7. Ciclul de confirmare oprire microprocesor	189
5.1.4. Conectarea dispozitivelor de memorie la magistralele unui sistem cu Z80 ..	189
5.1.4.1. Conectarea dispozitivelor ROM și SRAM	189
5.1.4.2. Conectarea memoriilor DRAM	191
5.1.5. Conectarea dispozitivelor I/E	193
5.1.5.1. Conectarea dispozitivelor I/E non-Z80	193
5.1.5.2. Conectarea dispozitivelor I/E din familia Z80	194
5.2. Programarea microprocesorului Z80	195
5.2.1. Formatul instrucțiunilor și modurile de adresare	195
5.2.2. Setul de instrucțiuni	197
5.2.2.1. Instrucțiuni de transfer	199
5.2.2.2. Instrucțiuni aritmetice	200
5.2.2.3. Instrucțiuni logice	202
5.2.2.4. Instrucțiuni de ramificare	204
5.2.2.5. Instrucțiuni de lucru cu stivă și de comandă a UCP	205
5.2.2.6. Instrucțiuni de intrare-ieșire	206
5.2.2.7. Codurile instrucțiunilor microprocesorului Z80	207
5.2.3. Tehnici de programare specifice sistemelor cu Z80	207
5.2.3.1. Utilizarea registrelor secundare	208
5.2.3.2. Operații pe blocuri de date	209
5.2.3.3. Utilizarea registrelor index	209

5.2.3.4. Utilizarea deplasărilor în operațiile aritmetice	210
5.3. Introducerea timpului ca variabilă în sistemele cu Z80	211
5.3.1. Circuitul CTC-Z80 (Counter Timer Circuit)	211
5.3.1.1. Modurile de lucru ale circuitului CTC-Z80	212
5.3.1.2. Programarea CTC-Z80	213
5.3.2. Organizarea sistemelor de introducere a timpului (SIT) cu CTC-Z80	215
5.3.3. Sisteme pentru întreruperi vectorizate cu CTC-Z80	217
5.4. Accesul direct la memorie în sistemele cu Z80	218
5.4.1. Structura internă a circuitului DMA-Z80	219
5.4.2. Programarea circuitului DMA-Z80	221
5.5. Interfațarea cu periferice în sistemele cu Z80	223
5.5.1. Circuitul de interfațare paralelă PIO-Z80	223
5.5.2. Modurile de lucru ale circuitului PIO-Z80	226
5.5.3. Programarea interfeței paralele PIO-Z80	230
5.6. Comunicația serială în sistemele cu Z80	230
5.6.1. Structura internă a circuitului SIO-Z80	230
5.6.2. Funcționarea circuitului SIO-Z80	233
5.6.2.1. Dialogul cu unitatea centrală de procesare	234
5.6.2.2. Posibilități de implementare a protocoalelor asincrone	235
5.6.2.3. Posibilități de implementare a protocoalelor sincrone	236
5.6.3. Programarea circuitului SIO-Z80	239
5.6.3.1. Registrele de comandă	240
5.6.3.2. Registrele de stare	244

Partea a III-a - Sisteme cu microprocesoare de 8 biți integrate

(microcontrolere)	249
-------------------------	-----

Cap.6. Familia de microcontrolere MCS-51 (8051)	251
6.1. Caracteristicile generale ale familiei MCS-51	251
6.2. Conexiunile externe ale familiei MCS-51	252
6.3. Arhitectura familiei MCS-51	254
6.3.1. Organizarea și configurarea memoriei	256
6.3.1.1. Memoria program	257
6.3.1.2. Memoria de date	258
6.3.2. Secvențierea operațiilor interne la MCS-51	261
6.3.2.1. Secvențierea instrucțiunilor de 1 ciclu mașină	261
6.3.2.2. Secvențierea instrucțiunilor de 2 cicluri mașină	262
6.3.2.3. Citirea și înscriserea datelor în memoria externă de date	264
6.3.3. Porturile de intrare-ieșire. Structură și operare	265
6.3.3.1. Configurația porturilor I/E	265
6.3.3.2. Facilitatea "citire-modificare-seriere" a porturilor	267
6.3.4. Funcții periferice integrate	267
6.3.4.1. Numărătoare-temporizatoare (Timer/Counters)	268
6.3.4.2. Interfața serială – moduri de operare	268
6.3.4.3. Comunicația multiprocesor	277
6.3.5. Sistemul de întreruperi	278
6.3.5.1. Funcționarea pas cu pas prin întreruperi	280
6.3.6. Resetarea microcontrolerelor familiei MCS-51	281
6.3.7. Regimuri de consum redus	282

6.3.7.1. Modul inactiv (Idle mode)	283
6.3.7.2. Modul deconectat (Power Down mode)	283
6.3.8. Programarea, verificarea și protecția memoriei EPROM	283
6.4. Facilitățile noi ale versiunilor perfecționate ale familiei MCS-51	288
6.4.1. Zona de numărătoare programabile (PCA)	289
6.4.2. Noile facilități ale Portului serial	293
6.4.2.1. Recunoașterea automată a adresei	293
6.4.2.2. Detecția erorilor de încadrare	294
6.5. Programarea familiei MCS-51	295
6.5.1. Setul de instrucțiuni	296
6.5.2. Programarea în limbaj de asamblare	305
6.5.3. Tehnici de programare specifice, pe grupe de instrucțiuni	309
6.5.4. Programarea perifericelor integrate	322
6.5.4.1. Programarea porturilor I/E	322
6.5.4.2. Programarea portului serial și a timerelor	322
Anexa I - Codurile instrucțiunilor microprocesoarelor 8080A și 8085A	325
Anexa II - Codurile instrucțiunilor microprocesorului Z80	326
Anexa III - Codurile instrucțiunilor familiei MCS-51	329
Bibliografie	331



MICROPROCESOARE ÎN CONDUCEREA AUTOMATĂ A PROCESELOR

Microprocesoarele, ca unități centrale de procesare a informației numerice, sunt utilizate în prezent atât pentru realizarea microcalculatoarelor de uz general, cât și în echipamentele de automatizare.

Primul domeniu a condus la o dezvoltare a microprocesoarelor în direcția obținerii de platforme de calcul cu facilități deosebite în ceea ce privește dimensiunea memoriei, puterea de calcul și viteza de operare. La aceste microcalculatoare, folosite pentru calcul numeric, datele de intrare constau dintr-un set de numere reale asupra cărora se operează în vederea producerii unui rezultat care constă dintr-un alt set de numere reale. Calculatorul execută programul o singură dată, obține rezultatul și se oprește la finalul execuției. Pentru a executa din nou același program este necesar să se introducă noi date, care sunt preluate fie prin intermediul unui periferic de intrare (spre exemplu tastatura), fie din anumite locații de memorie. În ceea ce privește legătura cu exteriorul, un sistem de calcul clasic este prevăzut cu un număr limitat de conexiuni necesare conectării de periferice standard intrare-ieșire (I/E): tastaturi, videoterminale, imprimante, memorii de masă etc.

Facilitățile oferite de microprocesoare le-au făcut deosebit de atractive și în domeniul automatizării instalațiilor și proceselor tehnologice. Spre deosebire de domeniul calculului clasic, în acest caz datele de intrare sunt informații provenite de la procesul fizic, obținute prin intermediul unei interfețe corespunzătoare. Datele prelevate din proces trebuie prelucrate în sistemul cu microprocesor în conformitate cu un algoritm numeric de conducere, iar rezultatele trebuie transmise în exterior spre elemente de execuție care asigură comanda procesului automatizat. Și în acest caz este necesară o interfață de adaptare a semnalelor numerice furnizate de microprocesor cu semnalele de forma (analogică sau discretă) și puterea solicitată de elementele de execuție.

Semnalele de intrare și de ieșire sunt primite și transmise de către microprocesor interfețelor de proces prin intermediul unor seturi de linii de intrare și ieșire, denumite *porturi de intrare/ieșire*.

Datele de intrare corespunzătoare mărimilor fizice măsurate se modifică în timp, adică nu rămân constante pe durata execuției programului. La un moment dat microprocesorul execută programul numai cu un set de date obținute de la portul de intrare și furnizează un set de date prin portul de ieșire. Odată ce rezultatele de la execuția precedentă sunt transmise spre procesul condus, microprocesorul extrage date noi de la portul de intrare și execută același program memorat. Secvența menționată mai sus poate fi executată de un număr de ori stabilit, poate rula un timp definit sau poate continua indefinit până când microprocesorul este resetat de un semnal extern.

Evoluția unui sistem cu microprocesor pentru conducere automată descrisă mai sus este reprezentată sintetic în fig. I.1 și caracterizează o funcționare în *timp real*. O astfel de funcționare impune necesitatea sincronizării procesului prelucrării electronice cu ritmul modificării informațiilor de intrare și, implicit, cu dinamica desfășurării procesului tehnologic. Rezultă că informația preluată din proces trebuie să fie prelucrată într-un interval limitat de timp, astfel ca rezultatul calculului să poată fi transmis în timp util procesului condus. Din acest motiv, la realizarea programelor de timp real, proiectantul trebuie să evalueze timpul necesar pentru desfășurarea fiecărei operații. Această estimare necesită cunoștințe atât despre *resursele fizice* cât și despre *resursele logice* ale sistemului cu microprocesor.

Pe de altă parte, un sistem cu microprocesor pentru conducere automată trebuie să poată modifica succesiunea unor programe (taskuri) de control, monitorizare și memorare, care folosesc aceleași interfețe hardware, succesiune care depinde de informațiile de intrare. Ca urmare, aceste sisteme trebuie să aibă abilitatea de a determina succesiunea optimă de execuție a diferitelor sarcini, într-un interval bine determinat de timp. O astfel de problemă se poate rezolva prin program de către un executiv de timp real, cu ajutorul sistemului de întreruperi și al ceasurilor. În plus, la microcalculatoarele pentru conducere automată este necesar un sistem I/E dezvoltat, precum și canale de acces direct la memorie pentru achiziția eficientă a informațiilor din procesul condus.

Deși sistemele cu microprocesoare au limitări impuse de lungimea cuvântului, dimensiunea memoriei și de un set restrâns de comenzi și instrucțiuni, aceste dispozitive pot asigura o alternativă economică în multe aplicații de control tradițional. Mai mult, microprocesoarele permit implementarea eficientă a conceptului de *conducere distribuită* a proceselor.

Specificul dezvoltării sistemelor bazate pe microprocesoare îl constituie interdependența aspectelor de proiectare hardware (HW) și software (SW).

Datorită legăturii puternice dintre HW și SW, proiectarea celor două componente necesită o interacțiune a celor două părți. De asemenea, intercondiționarea HW-SW are mai multe implicații asupra organizării sistemelor cu microprocesoare decât la proiectele HW convenționale.

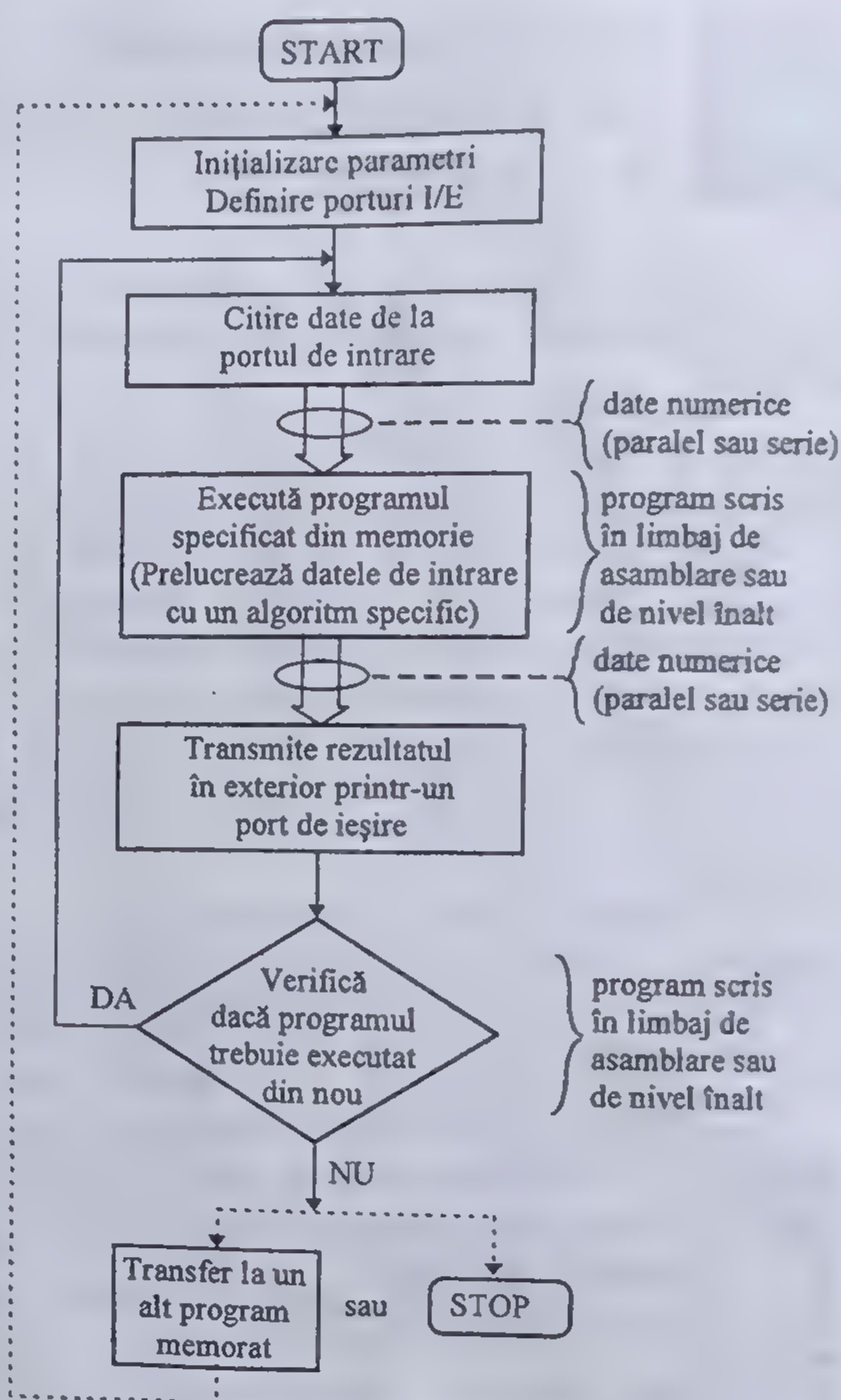


Fig.I.1. Secvențierea operațiilor specifice unei aplicații în timp real

1

Structura sistemelor cu microprocesoare

Din cele prezentate în partea introductivă au rezultat unele resurse specifice pe care trebuie să le aibă un sistem cu microprocesor orientat spre aplicații de timp real. În fig.1.1. este prezentată structura generală a unui astfel de sistem.

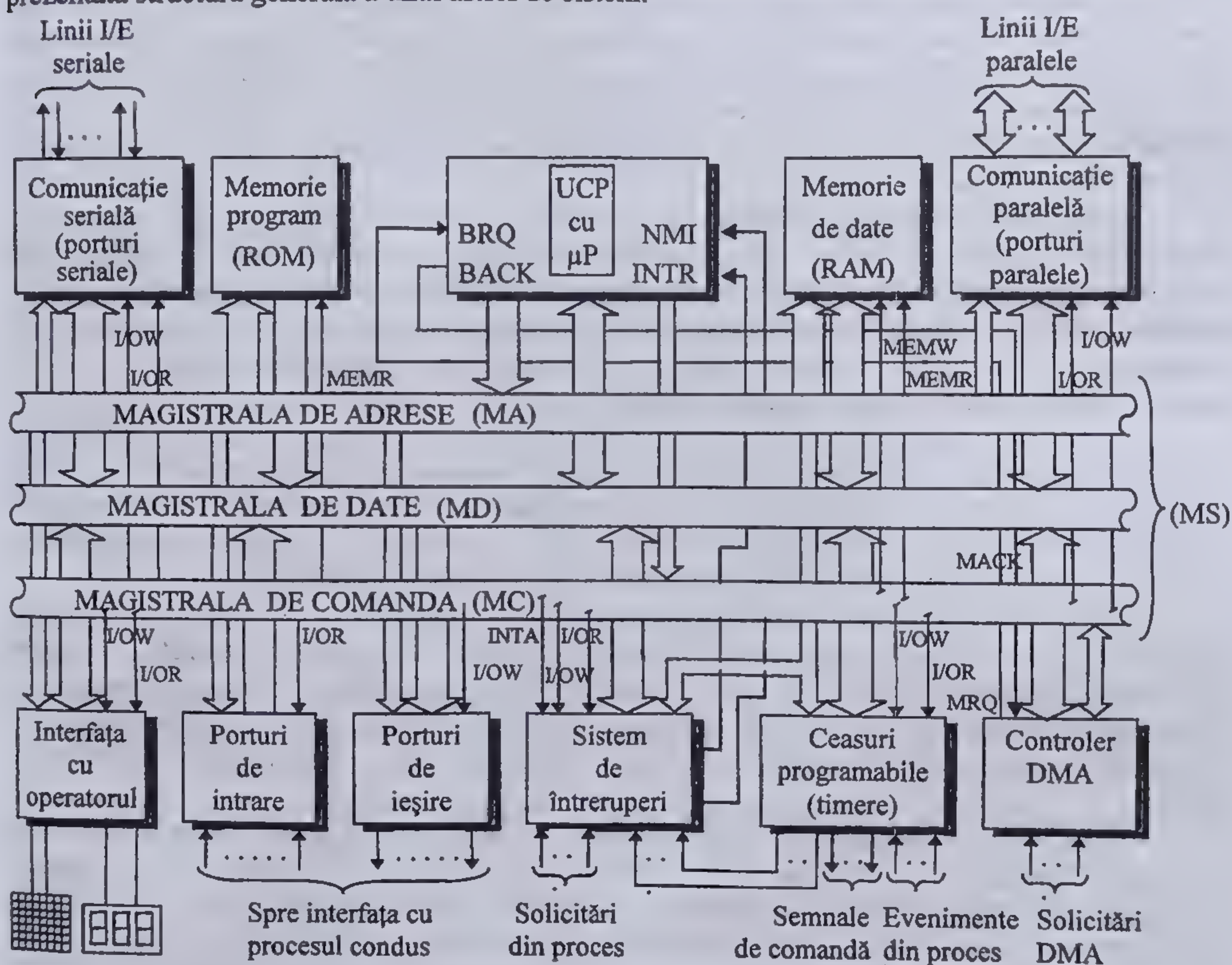


Fig.1.1. Configurația unui sistem cu microprocesor pentru conducere automată

Numărul și tipul elementelor folosite într-un sistem cu microprocesor depinde de specificul aplicației și pot varia în limite largi. În fig.1.1 sunt prezentate principalele blocuri funcționale întâlnite în structurile de conducere cu microprocesoare; o parte din aceste blocuri pot lipsi, de exemplu controlerul pentru acces direct la memorie sau comunicația paralelă.

Alături de *unitatea centrală de prelucrare* (UCP), *memoria* și *porturile I/E* sunt absolut necesare pentru funcționarea unui sistem cu microprocesor. Prin intermediul acestora, UCP primește sub formă de instrucțiuni comenzile necesare îndeplinirii unor acțiuni, precum și informațiile despre desfășurarea în timp a acestor acțiuni. Memoria trebuie să păstreze succesiunea comenzilor (programul), constantele și variabilele programului, iar porturile trebuie să permită actualizarea periodică a informațiilor și transmiterea de comenzi adecvate către procesul automatizat.

Atât memoria cât și porturile I/E trebuie să fie astfel constituite încât să poată dialoga cu UCP prin intermediul unei *magistrale de sistem* (MS) formată din *magistrala de adrese* (MA), *magistrala de date* (MD) și *magistrala de comandă* (MC), toate controlate de către microprocesor. În §1.3 se vor detalia elementele specifice care stau la baza unui dialog corect între UCP, memorie și porturile I/E. Aceste elemente se regăsesc și în cazul celorlalte circuite specializate care se conectează la MS prin intermediul unor porturi I/E incluse pe cip.

Îndeplinirea dezideratului de funcționare în timp real a sistemelor cu microprocesoare implică utilizarea unor *tehnici I/E* corespunzătoare. Esențială în acest caz este *tehnica întreruperilor*, care permite sincronizarea funcționării microprocesorului cu evenimente din procesul automatizat sau cu alte componente ale sistemului. Caracteristic sistemelor de conducere automată bazate pe microprocesoare este necesitatea unui *sistem de întreruperi* care să asigure degrevarea UCP de gestionarea unui număr important (de ordinul zecilor) de cereri din exterior. La fel de importantă este și existența unui *sistem pentru introducerea timpului* în prelucrarea informației, care utilizează dispozitive programabile specializate cunoscute sub numele de ceasuri programabile sau "timere".

O altă tehnică I/E folosită adesea în sistemele cu microprocesoare este *accesul direct la memorie* (DMA) care, prin intermediul unor microprocesoare specializate pentru transfer de date (controlere DMA), permite accelerarea transferurilor de informație între memorie și exterior. Astfel de situații se întâlnesc frecvent la conducerea proceselor cu o dinamică ridicată, ce comportă o achiziție de informație cu frecvență mare de eșantionare. Toate aceste probleme vor fi exemplificate prin studii de caz orientate pe principalele tipuri de microprocesoare.

Sistemele bazate pe microprocesoare pot fi concepute să dialogheze cu alte sisteme, fie în *mod paralel*, specific interfațării cu periferice aflate la distanțe mici, fie în *mod serie*, caracteristic comunicației la distanțe mari. În §2.1 și §2.2 sunt prezentate principalele noțiuni referitoare la interfațarea paralelă respectiv serială cu dispozitive I/E, iar în capitolul 3 sunt detaliate principiile *comunicației seriale*, mult utilizată în cazul sistemelor distribuite de conducere.

Succesul folosirii unui sistem cu microprocesor este direct influențat de interfața cu operatorul uman și care în conducerea unui proces poate lua forma specializată a consolei operatorului de proces (COP). De obicei această parte a sistemului corespunde interfațării cu dispozitive de intrare de tip matrice de taste și de ieșire de tip afișaj. Fiind destinată interfeței cu operatorul tehnolog, această consolă prezintă caracteristici specifice și trebuie tratată diferit de alte periferice.

Organizarea sistemului în jurul microprocesorului se face prin *agregarea* blocurilor funcționale îndeplinind condițiile de *compatibilitate fizică, informațională și logică*. Compatibilitatea fizică (hardware) este legată de standardizarea semnalelor de intrare și ieșire ale diferitelor blocuri din sistem. Compatibilitatea informațională se obține prin codificarea unică a datelor pentru toate blocurile sistemului. Compatibilitatea logică (software) implică posibilitatea execuției programelor elaborate pentru un model de către alte microsisteme având resurse similare.

Dacă resursele sistemului sunt controlate de un singur procesor (ca în fig.1.1) acesta este un sistem *monoprocessor*, iar dacă la resurse au acces mai multe procesoare se obține un sistem *multi(micro)procesor*. În ultimul caz este necesar și un sistem de arbitraj a accesului microprocesoarelor la resursele comune. Sistemele cu procesoare multiple sunt necesare în cazul automatizării proceselor complexe, prin distribuirea sarcinilor de conducere pe procesoare care funcționează în paralel și exploatează în comun o parte din resurse.

1.1. Componentele principale ale unui sistem cu microprocesor

Coordonatorul sistemului este *microprocesorul*, care de cele mai multe ori conține într-un singur cip toate elementele necesare unei unități centrale de procesare (UCP). Această unitate este

Capitolul 1 - Structura sistemelor cu microprocesoare

principalul element activ al sistemului și poate dialoga cu celelalte resurse (fig.1.1): memoria, porturile I/E și dispozitivele specializate pentru gestiunea întreruperilor, ceasurile, precum și cu controlerele pentru acces direct la memorie. În §1.1.1 se vor prezenta structura generală, conceptele de bază și regimurile de funcționare ale unui microprocesor, precum și elemente preliminare de realizare a programelor.

Memoria și porturile I/E sunt esențiale pentru funcționarea unui sistem cu microprocesor. Prin intermediul acestora UCP primește, sub formă de instrucțiuni, comenzile necesare îndeplinirii unor acțiuni precum și informațiile despre desfășurarea reală a acțiunilor. Memoria trebuie să păstreze succesiunea comenzilor (programul), iar porturile să permită actualizarea periodică a informațiilor și să transmită comenzi adecvate spre procesul condus. Atât memoria cât și porturile I/E trebuie să fie astfel constituite încât să poată dialoga cu UCP prin intermediul magistralei sistem (MS), formată din liniile magistralelor de adrese (MA), de date (MD) și respectiv de comandă (MC), controlate de microprocesor. În §1.1.2 se vor detalia elementele specifice care stau la baza unui dialog corespunzător între UCP și memorie iar în §1.1.3 între UCP și porturile I/E.

1.1.1. Microprocesorul

Microprocesorul este elementul definitoriu al sistemului, atât prin resursele fizice interne și setul de instrucțiuni acceptat, cât și prin semnalele generate în exterior prin intermediul cărora comunică cu celelalte dispozitive componente. El este coordonatorul (master) tuturor activităților desfășurate în sistem, având inițiativa în relațiile cu celelalte componente ale sistemului, care sunt elemente pasive (slave).

Din punct de vedere funcțional, un microprocesor este un sistem secvențial sincron complex, ce poate fi caracterizat prin modelul abstract de automat finit:

$$A = (I, S, O, f, g, s_0),$$

în care: I, S, O sunt mulțimi finite de intrare, stare și ieșire; $f: I \times S \rightarrow S$ și $g: I \times S \rightarrow O$ - sunt funcțiile de tranziție și respectiv de ieșire, iar s_0 - este starea inițială. Funcția de tranziție caracterizează evoluția internă a sistemului, iar funcția de ieșire evoluția semnalelor de ieșire. Microprocesoarele sunt automate deterministe, având $|f(i, s)| = 1$ și $|g(i, s)| = 1$, $\forall i \in I, \forall s \in S$, adică au o evoluție unică pentru o pereche (i, s) dată. Pentru a se putea asigura o evoluție previzibilă este necesar ca automatul să poată fi adus (întotdeauna) într-o stare inițială s_0 .

Realizarea fizică a microprocesoarelor a plecat de la sinteza structurilor secvențiale, urmărind materializarea unor unități centrale de procesare integrate. De aceea, se poate spune că un microprocesor este un procesor delimitat funcțional și constructiv, comparabil ca structură și posibilități cu unitatea centrală a unui calculator numeric clasic. Poate fi realizat într-unul sau mai multe circuite LSI și este destinat prelucrării informației cu ajutorul unei unități logico-aritmetice (ULA), a registrelor interne (R) și a unei unități de comandă (UC).

Deși fiecare tip de microprocesor are o structură internă specifică, având în vedere cele de mai sus, se poate imagina o schemă simplificată a acestor dispozitive, ca în fig.1.2.

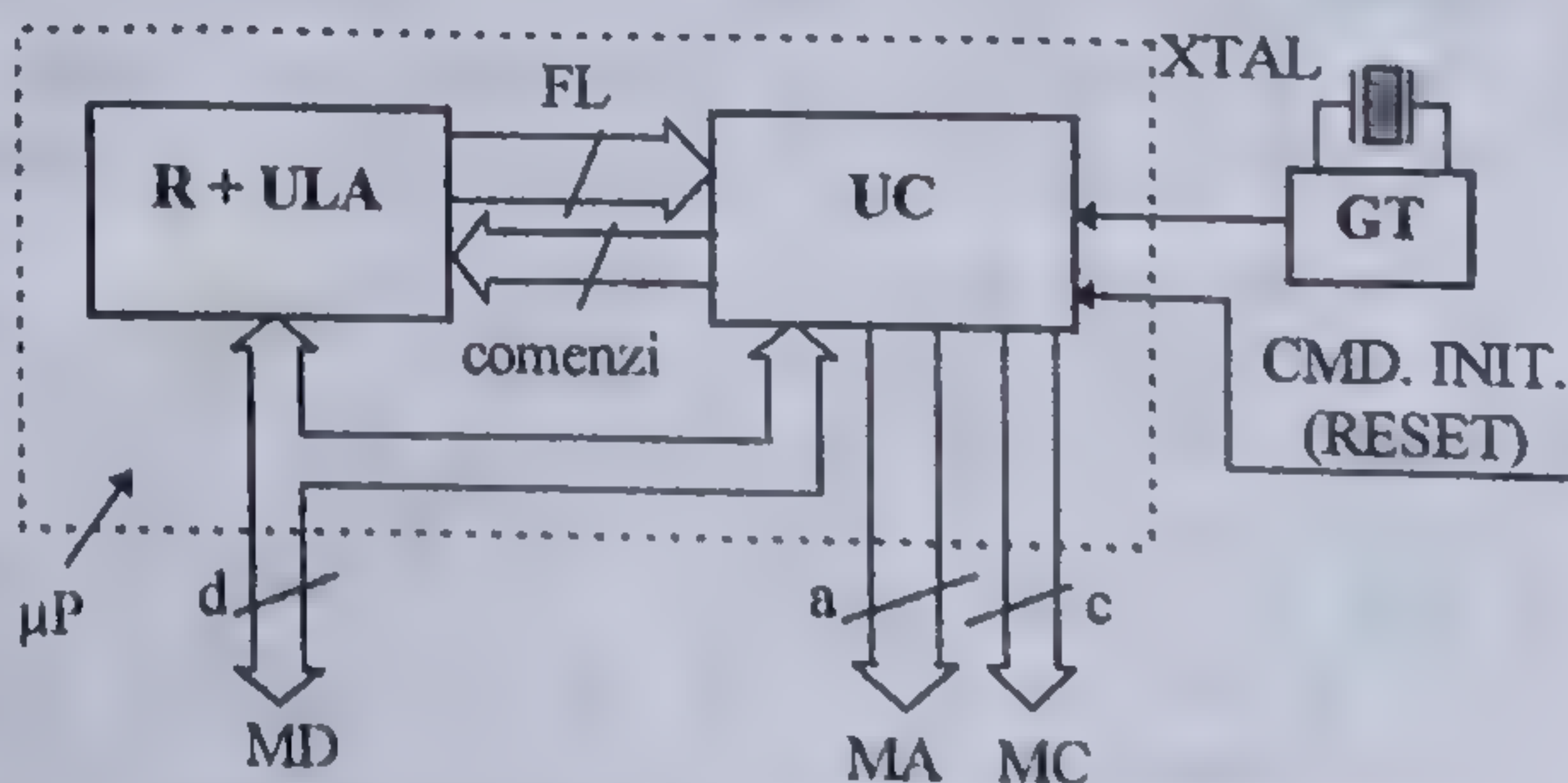


Fig.1.2. Structura internă simplificată a unui microprocesor

Registrele R au rolul de a memora temporar cuvintele ce urmează a fi prelucrate în ULA. Numărul acestor registre poate fi la limită zero, toate memorările realizându-se într-o memorie externă. Deși ar simplifica structura microprocesorului, o astfel de soluție nu este avantajoasă deoarece ar reduce considerabil viteza de execuție prin apelarea repetată la memoria externă. Din

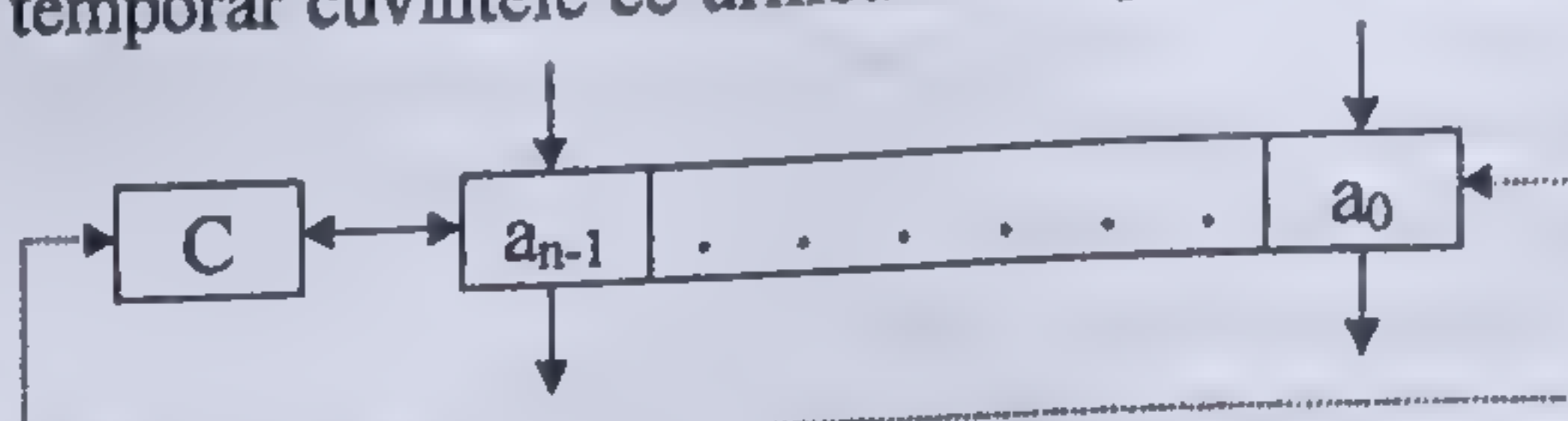


Fig.1.3. Schema unui registru acumulator cu o capacitate de n biți

acest motiv, la microprocesoarele actuale blocul R conține un număr de registre care asigură intern un transfer eficient și o creștere apreciabilă a vitezei de execuție a instrucțiunilor. Numărul acestor registre variază de la o realizare la alta, crescând apreciabil la microprocesoarele evolute. Un registru special existent la microprocesoarele de uz general este *registru acumulator*, care are rolul de a păstra rezultatul operării logice/aritmetice sau în care se depune un cuvânt pentru/de la un port I/E. Notat cu A, un astfel de registru are toate facilitățile de intrare/ieșire: paralelă și serie, precum și de deplasare dreapta/stânga (fig.1.3).

Obișnuit, acestui registru i se atașează un circuit basculant bistabil în care se pot transfera și testa biții a_{n-1}, \dots, a_0 ai cuvântului sau care poate indica o depășire/împrumut la operațiile aritmetice. Este notat cu C (Carry) și conținutul lui poate fi utilizat în operare.

Unitatea logico-aritmetică (ULA) realizează prelucrarea logică sau aritmetică a operanzilor sub formă de cuvinte binare. *Lungimea cuvântului (l_C)* cu care operează ULA este un criteriu de clasificare a microprocesoarelor.

Uzual sunt realizate microprocesoare având $l_C = 1, 4, 8, 16, 32$ sau 64 de biți. Natural, microprocesoarele cu dimensiune redusă a cuvântului au capacitate modestă de prelucrare a informației numerice, dar pentru aplicații restrânse pot fi avantajoase. Astfel, pentru realizarea *automatelor programabile* cu prelucrare la nivel de bit sunt ideale microprocesoarele monobit [18, 27], iar pentru comanda unor instalații cu motoare pas cu pas sunt avantajoase cele de 4 biți [30, 34]. La realizarea sistemelor pentru conducerea automată se utilizează în mod curent microprocesoarele de 8 și 16 biți [11, 17, 19, 21, 23], iar cele de 32 și 64 de biți se folosesc mai mult în structuri de tip microcalculator, dar și în structurile numerice moderne de control.

Operarea logică se realizează bit cu bit, în conformitate cu regulile de prelucrare din algebra Boole.

Fie $B = (b_{n-1} b_{n-2} \dots b_1 b_0)$ și $C = (c_{n-1} c_{n-2} \dots c_1 c_0)$ două cuvinte binare de n biți, existente în două registre interne, dintre care unul poate fi chiar acumulatorul. Rezultatul unei operării logice $A = B \square C = (a_{n-1} a_{n-2} \dots a_1 a_0)$ este tot un cuvânt de n biți, unde $a_i = b_i \square c_i$; $a_i, b_i, c_i \in \{0, 1\}$, $i = 0, n-1$. Obișnuit, la microprocesoarele uzuale operatorul \square poate fi \vee, \wedge, \oplus sau \neg , în ultimul caz fiind aplicat asupra unui singur operand.

Operarea aritmetică se realizează în baza 2, având la microprocesoare operațiile: $+, -, *, /$ și poate conduce la biți de depășire sau împrumut. Din acest motiv, ULA are atașate un număr de CBB-uri care să poată memora temporar condițiile de prelucrare a informației și, eventual, să fie utilizate în operarea ulterioară. Aceste CBB-uri se numesc *indicatori de condiție* sau *fanioane* (Flags) și sunt grupate într-un registru. Principalii indicatori folosiți la microprocesoare sunt: C (Carry), S (Sign), P (Parity), precum și Z (Zero). Conținutul registrului A și al fanioanelor definesc *programului* (Program Status Word).

Blocul de comandă (UC) realizează întreaga coordonare a evoluției microprocesorului prin recunoașterea și decodificarea instrucțiunilor și generarea comenzilor în vederea execuției acestora: în interior spre R+ALU și în exterior prin magistralele de adresă (MA), de date (MD) și de comandă (MC).

Capitolul 1 - Structura sistemelor cu microprocesoare

Toate operațiile din μP se execută sincron cu semnalele furnizate de un *generator de tact* (GT). Obişnuit, acest generator conține un oscilator pilotat de un cristal de cuarț (XTAL), pentru a asigura o stabilitate înaltă ciclurilor interne și un formator de semnale de tact. Semnalul de tact poate fi cu două faze (Φ_1 și Φ_2) sau cu o fază (Φ), în funcție de tipul microprocesorului. De regulă, la primele generații de microprocesoare se folosesc generatoare de tact cu două faze (fig.1.4a), iar la următoarele cu o singură fază (fig.1.4b).

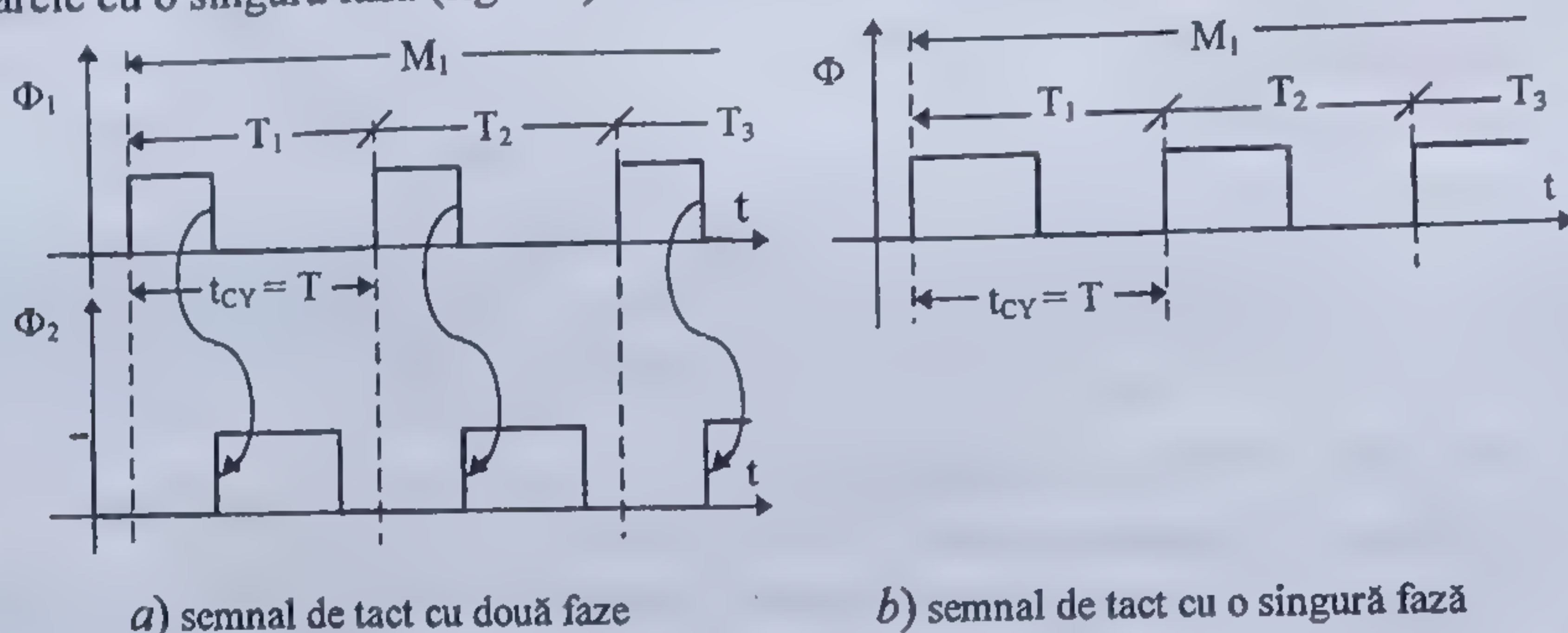


Fig.1.4. Semnale de tact pentru microprocesoare

Proiectantul de sistem trebuie să țină cont de acest aspect și să adopte o structură de GT corespunzătoare sau să o proiecteze în conformitate cu specificațiile de firmă. Fiecare producător indică durata posibilă a unui ciclu (t_{CY}) și în funcție de aceasta se alege și frecvența de rezonanță a cristalului de cuarț.

1.1.1.1. Stare mașină. Ciclu mașină. Ciclu Instrucțiune

Unitatea elementară de timp la care se face referire în funcționarea unui microprocesor este *durata unui ciclu de tact* - t_{CY} . Plecând de la acest element se pot defini principalele noțiuni specifice funcționării microprocesoarelor.

Starea mașină (T) este durata de timp în care are loc cea mai simplă operație internă într-un microprocesor. O stare mașină este delimitată de durata unei perioade de tact și se notează cu $T_i = t_{CY}$ ($i = 1, 2, 3, \dots$). Aceste perioade specifică mulțimea stărilor între care evoluează un microprocesor ca automat finit și care este fixată în memoria internă a UC în corespondență cu setul de instrucțiuni cu care este dotat. Din acest punct de vedere, un microprocesor poate fi considerat un *automat algoritmic de stare* (algorithmic state machine) care dispune de un set fix de instrucțiuni.

Ciclu mașină (M) este o succesiune finită de stări mașină (T_1, T_2, \dots, T_k), pe durata cărora se execută operații fundamentale de schimb de informații în interiorul microprocesorului sau între microprocesor și exterior (memorie sau porturi I/E). Exemple de astfel de cicluri: transfer de informații între registrele interne, citire din memorie, scriere în memorie, citire dintr-un port de intrare, scriere într-un port de ieșire. În corespondență cu tipul unui ciclu mașină, un microprocesor va avea nevoie pentru îndeplinirea unei operații fundamentale de un anumit număr de stări mașină:

$$M = \sum_{i=1}^k T_i.$$

Un ciclu mașină esențial funcționării unui microprocesor este *ciclu de aducere a instrucțiunii din memorie* (Instruction Fetch Cycle). În acest ciclu, de tip citire din memorie, se obțin principalele informații despre instrucțiunea care urmează să fie executată. Din acest motiv,

execuția unei instrucțiuni începe întotdeauna cu un ciclu fetch, care este denumit și *ciclu mașină 1* (Machine Cycle One) și este notat prin M1.

Ciclul instrucțiune (CI) reprezintă succesiunea de cicluri mașină M_j pe care trebuie să le parcurgă un microprocesor în vederea execuției complete a unei instrucțiuni. Deci, durata unui ciclu instrucțiune se poate stabili prin următoarea relație:

$$CI = \sum_{j=1}^l M_j = \sum_{j=1}^l \sum_{i=1}^k T_{ij}.$$

Prin urmare, se poate calcula cu precizie durata de execuție a unui program, care constituie o mărime importantă în funcționarea în timp real. Cele mai simple instrucțiuni sunt cele de transfer între registre interne și au loc într-un singur ciclu mașină, care este chiar ciclul fetch. Instrucțiunile care impun dialogul cu exteriorul (memorie sau porturi I/E) au mai multe cicluri mașină, primul fiind întotdeauna un ciclu de tip M1.

Inițializarea microprocesorului

Fiind o mașină de stare complexă, este necesar ca evoluția microprocesorului să poată fi totdeauna controlabilă. Pentru aceasta, automatul trebuie adus într-o *stare inițială* (s_0) înaintea aplicării unor comenzi la intrare (teorema stării inițiale) [16]. Numai în acest mod microprocesorul va avea o funcționare previzibilă, dacă nu apar defecte HW sau erori de program (defecte SW). Chiar în cazul existenței unor astfel de defecțiuni, aplicarea unor proceduri de testare impune necesitatea aducerii mașinii într-o stare inițială.

La toate microprocesoarele se folosește o inițializare HW prin aplicarea unei comenzi externe (CMD. INIT. în fig.1.2). O astfel de comandă asigură forțarea unor registre vitale ale microprocesorului într-o stare bine definită, fapt ce a condus la consacrarea acestei comenzi prin denumirea RESET. Comanda de inițializare trebuie să se activeze automat ori de câte ori se realizează conectarea microprocesorului la sursa de alimentare sau la cererea operatorului.

Obișnuit, se folosește o schemă simplă ca aceea din fig.1.5, care asigură printr-un grup RC durata impusă de producător pentru semnalul RESET. Deoarece microprocesorul acceptă semnale (de cele mai multe ori de tip TTL) cu anumite restricții privind durata fronturilor, este necesară o formare printr-un trigger Schmitt (T.S.). Multe microprocesoare au astfel de formatoare incluse pe cip.

Trebuie menționat faptul că unele tipuri de microprocesoare, în special cele orientate spre aplicații de control (microcontrolere) dispun și de facilități interne de inițializare HW, prin care se generează o comandă de resetare în anumite condiții de execuție a programului.

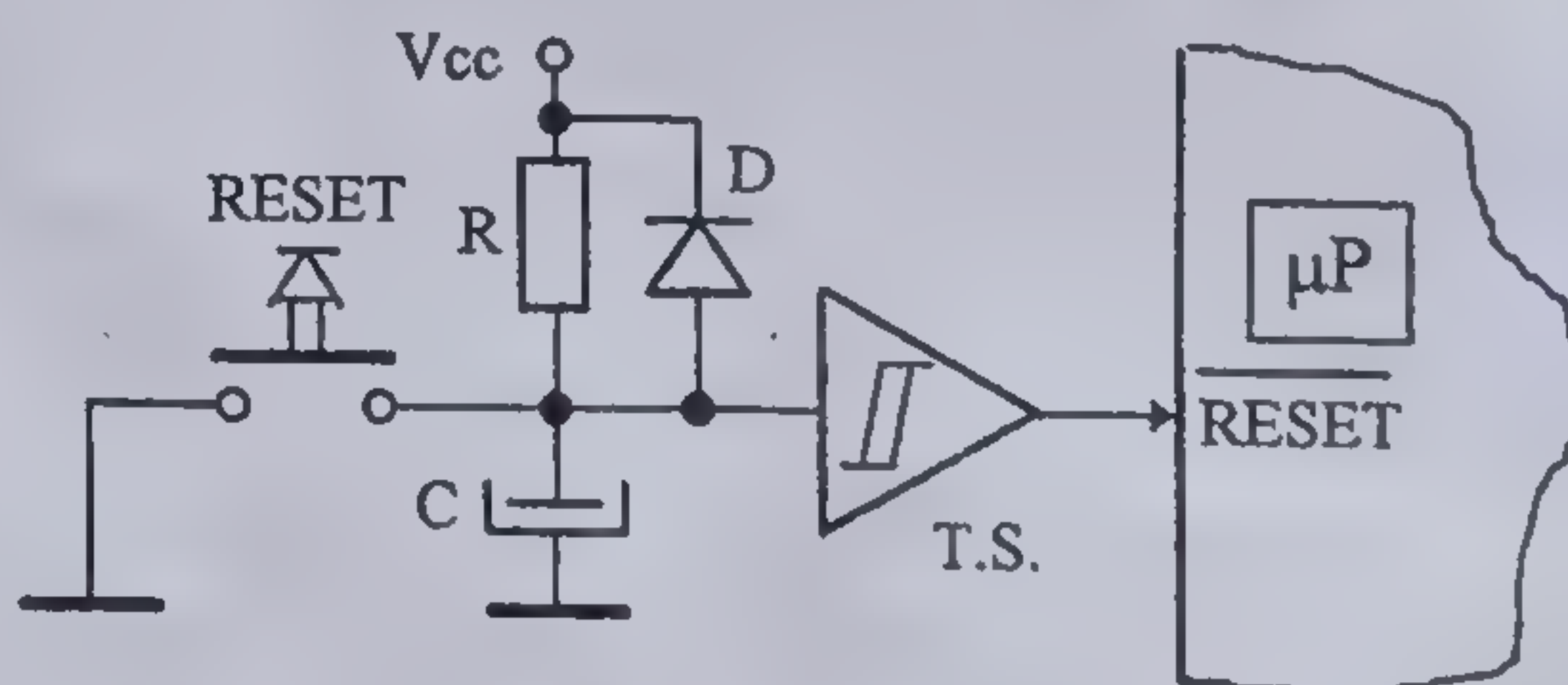


Fig.1.5. Schema de inițializare a unui microprocesor

1.1.1.2. Magistralele unui microprocesor

Pentru dialogul cu dispozitivele externe, microprocesoarele actuale sunt dotate cu trei seturi de linii care permit adresarea lor, transferul de date sau instrucțiuni și controlul transferului. Aceste grupuri de linii sunt denumite *magistrale*: de adrese (MA), de date (MD) și de comandă (MC) - fig.1.2 și formează *magistrala sistemului* (MS - în fig.1.1). În jurul acestei magistrale se organizează un sistem cu microprocesor.

Capitolul 1 - Structura sistemelor cu microprocesoare

Magistrala de adrese este *unidirecțională*, de la microprocesor spre exterior și conține a linii fizice. Acest număr definește dimensiunea spațiului de adresare directă a microprocesorului, care cuprinde 2^a adrese distincte. Prin intermediul liniilor de adresă microprocesorul specifică o locație de memorie sau un port I/E. La unele dintre microprocesoarele actuale, prin instrucțiuni speciale, este accesibil un spațiu de adresare directă a porturilor I/E, cu folosirea unui număr mai restrâns de linii de adresă. De exemplu, multe microprocesoare care au $a=16$, pot specifica prin 8 biți un spațiu I/E pentru maximum 256 porturi de intrare și maximum 256 porturi de ieșire. Deși o astfel de soluție este eficientă, uneori acest spațiu nu oferă suficientă flexibilitate tocmai datorită numărului redus de instrucțiuni specifice. În acest caz se poate folosi spațiul de adresare al memoriei, când porturile I/E sunt adresate ca locații de memorie ("memory mapped I/O") și beneficiază de instrucțiunile de dialog cu memoria, mult mai numeroase decât cele dedicate spațiului I/E. Evident, ultima posibilitate se poate aplica numai atunci când spațiul de adresare al memoriei nu este complet utilizat, ceea ce se întâmplă curent în cazul sistemelor pentru conducere automată.

Magistrala de date este *bidirecțională* și prin lărgimea sa, d , se specifică dimensiunea cuvântului care poate fi transferat la un moment dat de către microprocesor. Prin intermediul magistralei de date microprocesorul acceptă atât instrucțiuni cât și date și furnizează în exterior rezultate ale prelucrării interne. Din acest motiv este necesară bidirecționalitatea acesteia. Microprocesorul controlează sensul de transfer pe MD în corespondență cu tipul operației care urmează a fi efectuată. La *citire*, MD are liniile orientate spre intrare, iar la *scriere* MD este orientată spre ieșire. Asociat cu orientarea magistralei de date, microprocesoarele generează și semnale distincte care facilitează proiectarea logicii de amplificare a acestei magistrale.

Magistrala de comandă este constituită din linii care permit microprocesorului să realizeze dialogul cu dispozitivele externe: memoria și porturile I/E. Din acest motiv, toate microprocesoarele trebuie să genereze cel puțin următoarele patru tipuri de semnale de comandă (v.fig.1.1):

- citire din memorie (MEMR - Memory Read)
- scriere în memorie (MEMW - Memory Write)
- citire dintr-un port (I/OR - Input/Output Read)
- scriere într-un port (I/OW - Input/Output Write)

Aceste semnale sunt generice, fiecare tip de microprocesor furnizând un număr specific de semnale de comandă, care însă pot fi decodificate și generate sub forma semnalelor enumerate mai sus.

1.1.1.3. Regimuri de funcționare ale microprocesoarelor

În funcționarea sistemelor cu microprocesoare intervin situații care necesită evoluții diferite de cea specifică execuției normale a programului. Astfel de situații apar în cazul în care microprocesorul trebuie să dialogheze cu memorii mai lente, atunci când resursele sistemului trebuie să fie accesate de un alt coordonator sau în cazul în care este necesară abandonarea temporară a programului în curs de execuție. Pentru a putea satisface astfel de situații, microprocesoarele pot avea următoarele regimuri de funcționare:

a) **Regimul de execuție normală a programului.** După inițializare microprocesorul depune pe MA adresa primei instrucțiuni a programului. Adresa este conținută într-un registru numit *numărător (contor) de program* - PC (Program Counter). Apoi, contorul de program se incrementează, pregătind următoarea adresă. După aducerea unei instrucțiuni în registrele interne, UC (fig.1.2) o decodifică și generează succesiunea de microoperații în vederea execuției (numărul de cicluri mașină și tipul acestora). În același timp UC generează semnale de comandă în interior, spre R+ULA sau în exterior, pe MC. La terminarea unei instrucțiuni se trece la următoarea,

ș.a.m.d., urmând o execuție secvențială a programului. La multe microprocesoare registrul PC are dimensiunea identică cu lărgimea a a magistralei de adresă. La fiecare resetare acesta se inițializează cu adresa de debut a programului.

b) **Regimul de așteptare (WAIT).** Este necesar pentru sincronizarea microprocesorului cu dispozitive externe mai lente. Pentru astfel de situații microprocesoarele dispun de linii de comandă care asigură, la cererea dispozitivului selectat, includerea unor stări mașină inactive, numite *stări de așteptare* și notate T_{WAIT} . Cel mai adesea astfel de situații apar în cazul folosirii unor cipuri de memorie cu un timp de răspuns mai mare decât durata unui ciclu procesor (t_{CY}). Deoarece dispozitivele de memorie au un rol pasiv, în astfel de situații rezolvarea sincronizării cade în sarcina proiectantului de sistem, care trebuie să introducă o structură suplimentară denumită *logică de așteptare*. Spre exemplu, la citirea memoriilor lente între microprocesor și memorie, principal, trebuie să existe următorul dialog (fig.1.6):

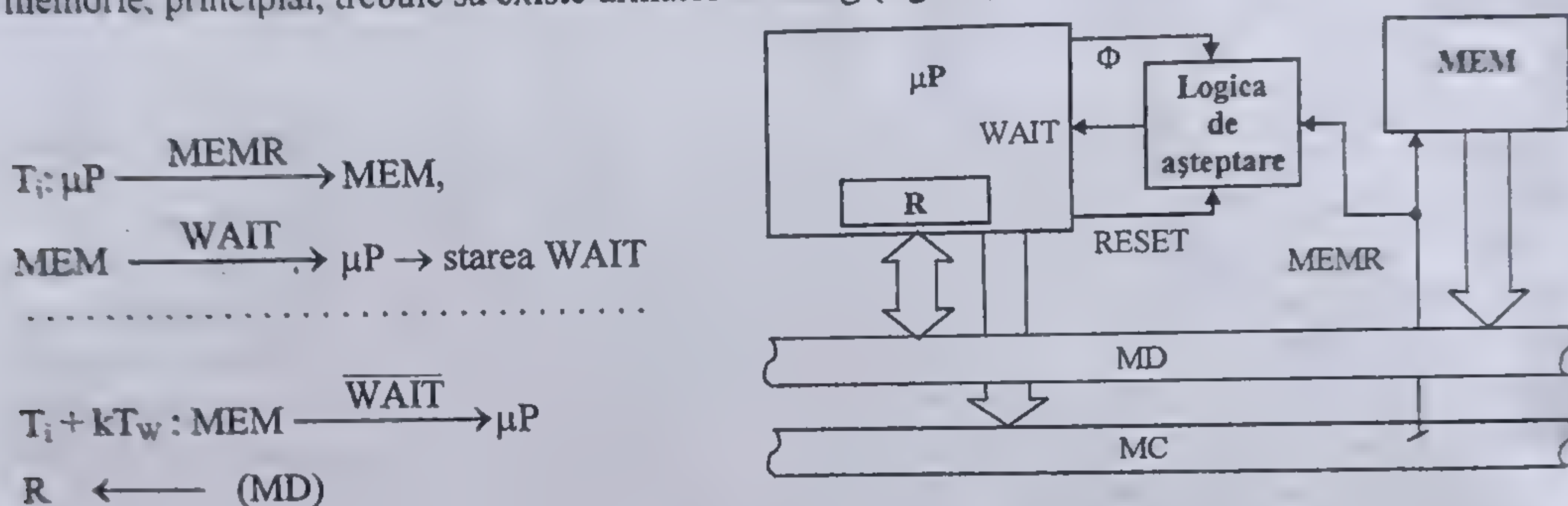


Fig.1.6. Principiul implementării regimului de așteptare

Semnalul MEMR declanșează trecerea microprocesorului în regimul de așteptare. Pe întreaga durată a acestui regim microprocesorul menține activ semnalul MEMR, astfel încât, după un număr k de cicluri T_w , pe MD va exista informația corectă. După terminarea perioadei de așteptare ($WAIT=1$), microprocesorul transferă informația de pe MD într-un registru R.

Pentru realizarea blocului logică de așteptare din fig.1.6 este necesar să se cunoască numărul k de stări T_w și să se realizeze o sincronizare cu semnalul de tact al microprocesorului (Φ). De asemenea, este necesară inițializarea acestui bloc odată cu microprocesorul. Cel mai simplu această tehnică se poate realiza cu un lanț de CBB-uri sincrone de tip D.

Este evident faptul că trecerea microprocesorului în regimul de așteptare reduce considerabil viteza de execuție a sistemului și complică structura acestuia. Din acest motiv este mai avantajos să se folosească memorii cu timp de răspuns corespunzător.

c) **Regimul de cedare de magistrale.** Se realizează prin trecerea celor trei magistrale ale microprocesorului în starea de înaltă impedanță. Se folosește pentru *accesul direct la memorie* (Direct Memory Access). La acest regim se apelează atunci când dispozitive externe de mare viteză (de exemplu convertoare A/D rapide) trebuie să aibă acces la locații de memorie sau când se transferă un bloc mare de date de la o memorie de masă (unități de discuri magnetice dure sau flexibile). De asemenea, regimul de cedare de magistrale este necesar realizării structurilor multimicroprocesor. Practic, acest regim implică transferul coordonării resurselor sistemului unui alt coordonator decât microprocesorul UCP.

Microprocesoarele actuale sunt prevăzute cu linii de comandă specifice (vezi fig.1.1): de solicitare a magistralelor BRQ (Bus ReQuest) și de acceptare BACK (Bus ACKnowledge). Prin intermediul acestor linii, solicitantul de MS poartă un dialog cu confirmare ("handshaking") cu microprocesorul unității centrale de procesare. Dacă solicitarea este acceptată, UCP își suspendă

Capitolul 1 - Structura sistemelor cu microprocesoare

activitatea, permițând noului coordonator să controleze resursele sistemului. În momentul în care acesta și-a încheiat activitatea pentru care a solicitat MS, microprocesorul din UCP își reia funcționarea din punctul în care a suspendat-o.

d) **Regimul de întreruperi.** Permite tratarea prioritară a unor solicitări externe prin întreruperea temporară a rulării normale a programului principal și execuția unei rutine asociate solicitantului de întrerupere. În acest mod, un sistem cu microprocesor poate realiza un răspuns în timp real la diferite evenimente apărute în procesul condus. Toate microprocesoarele sunt dotate cu cel puțin o linie de intrare pentru solicitarea de întreruperi. Obişnuit, există două tipuri de întreruperi: *nemascabile* și *mascabile* (NMI- Non Maskable Interrupt și INTR - în fig.1.1).

O întrerupere nemascabilă are două caracteristici definitorii: nu poate fi inhibată prin program și rutina de servire a întreruperii are o localizare prestabilită, definită de producătorul microprocesorului. Din cauza acestor caracteristici, NMI se folosește pentru controlul unor situații deosebite, cum ar fi funcționarea sistemului la iminența dispariției tensiunii de alimentare.

Întreruperile mascabile au prioritate mai redusă în tratare decât cele nemascabile și se caracterizează prin faptul că pot fi invalidate (mascate) prin program. Instrucțiuni speciale de comandă asigură setarea/resetarea unui fanion de întreruperi (Interrupt Flag). Întreruperile mascabile reprezintă categoria cea mai răspândită de întreruperi folosite la microprocesoare. Pentru acceptarea unei întreruperi mascabile, între solicitantul de întrerupere și microprocesor are loc un dialog cu confirmare. Cererea de întrerupere este primită de microprocesor pe linia dedicată INTR, după care acesta confirmă acceptarea printr-un semnal de achitare INTA (INTerrupt Acknowledge) transmis prin MC (în fig.1.1) sau printr-o linie dedicată.

În sistemele de conducere automată există mai multe surse de întrerupere (din sistem și din procesul condus) decât liniile de întrerupere pe care le posedă un microprocesor. Din acest motiv se prevăd *sisteme de întreruperi* (v. și fig.1.1) bazate pe circuite LSI specializate, care asigură gestionarea și tratarea ordonată a solicitărilor multiple. Realizarea unei astfel de structuri cade în sarcina proiectantului de sistem, care trebuie să rezolve atât probleme HW cât și SW specifice. Importanța acestei probleme a făcut ca la unele microprocesoare (de exemplu la microcontrolere) să fie dezvoltate pe același cip sisteme puternice de întreruperi, care elimină necesitatea implementării lor în exterior.

1.1.1.4. Programarea microprocesoarelor

După cum s-a mai menționat, un microprocesor poate efectua numai sarcinile care îi sunt transmise printr-un *program* alcătuit dintr-o succesiune de instrucțiuni. Funcționarea secvențială a microprocesoarelor necesită specificarea pas cu pas a operațiilor care trebuie executate, fapt ce impune transpunerea problemelor de rezolvat într-un *algorithm*. Corespunzător algoritmului și operațiilor fundamentale pe care le poate executa un microprocesor, se poate stabili succesiunea necesară a instrucțiunilor.

Microprocesoarele nu pot executa decât instrucțiunile exprimate sub formă binară. Un program în care instrucțiunile sunt descrise sub *formă numerică*, prin cuvinte binare, este numit program în *limbaj mașină*. O astfel de exprimare este anevoioasă și inadecvată pentru programator. În vederea depășirii acestor dificultăți s-a încercat codificarea instrucțiunilor sub alte forme, mai accesibile și mai ușor de manipulat. Primul pas l-a constituit folosirea codurilor numerice mai puternice, cum ar fi codurile octal sau hexazecimal. Deși mai concisă, nici această formă de reprezentare a instrucțiunilor nu este adecvată. De aceea au fost create limbaje artificiale, în care instrucțiunile sunt exprimate sub *formă simbolică*, denumite *limbaje de programare*. În funcție de raportul în care se află cu limbajul mașină și cu limbajul natural al omului, limbajele de programare se împart în două categorii: limbaje de nivel coborât, reprezentate în principal de *limbajul de asamblare* și limbaje de nivel înalt.

1) Limbaajul de asamblare, ca și limbajul mașină, este specific fiecărui tip de microprocesor. Fiecărui tip de instrucțiune în limbaj mașină i se atașează o expresie simbolică, care abreviază exprimarea sa în limba engleză. Această expresie, denumită *mnemonica instrucțiunii*, include prescurtarea numelui operației (*mnemonica operației*) și numele operanzilor (dacă aceștia există). De exemplu, instrucțiunea **MVI A, data** a microprocesoarelor de 8 biți ale firmei Intel provine de la: *MoVe Immediate to Accumulator data*, desemnând operația de transfer în acumulator a octetului *data* dispus în memorie imediat după codul operației. Prin folosirea mnemonicelor, programatorul poate scrie *programul sursă* cu mai multă ușurință decât într-un cod numeric.

Pentru a putea fi executat de către microprocesor, programul sursă scris în limbaj de asamblare trebuie convertit (translatat) în limbajul mașinii. Dacă traducerea se face de către programator (în binar sau în hexazecimal) se spune că se obține codul obiect *asamblat manual*. Acest lucru se poate face de obicei doar pentru secvențe scurte de instrucțiuni. Pentru programe mai complexe se apelează la un program specializat de *asamblare automată*, denumit *asamblor*, care este realizat, de obicei, de firma care produce și microprocesorul. Există însă și *cross-asamblatoare*, adică programe care îndeplinesc aceeași funcție ca și asamblorul, dar rulează pe un alt tip de microprocesor decât cel pentru care se generează codul mașină (numit microprocesor țintă).

În orice caz, un program de tip asamblor trebuie în primul rând să recunoască setul de instrucțiuni al microprocesorului țintă. Fiecare instrucțiune în limbaj de asamblare este translatată într-o singură instrucțiune în cod mașină, care urmează a fi executată de către microprocesorul țintă. De aceea, pentru asamblor, aceste instrucțiuni se mai numesc și *instrucțiuni executabile*.

Dar, acest lucru nu este suficient pentru a putea scrie programe complexe în limbaj de asamblare. Este necesar ca asamblorul să definească un grup de reguli sintactice de scriere a fișierului sursă, precum și un număr (relativ restrâns) de instrucțiuni pentru care nu se generează cod mașină, denumite *directive de asamblare* și care au rolul de interfață cu programatorul. Pentru asamblor, acestea sunt instrucțiuni *neexecutabile* sau *pseudoinstrucțiuni*. Ele sunt puse la dispoziția programatorului pentru a-l ajuta să-și organizeze programul sursă: să definească modulele de program și relațiile dintre ele, să definească și/sau să declare datele programului (constante sau variabile), să structureze programul astfel încât să-l facă mai lizibil și ușor de înțeles, pseudoinstrucțiunile fiind interpretate numai de programul asamblor. Exemple:

ORG exp (ORiGine) - permite stabilirea adresei de debut a unui modul de program (exp)

DB 45h (Define Byte) - definește locul de amplasare a unei variabile de 1 octet, pe care o și inițializează cu 45h.

Unele programe de asamblare, numite *macroasamblatoare*, extind posibilitățile unui asamblor obișnuit. Ele permit definirea de către programator de noi "instrucțiuni", sub forma unor secvențe de instrucțiuni executabile și/sau neexecutabile numite *macroinstrucțiuni*.

Codul generat de asamblor poate fi *cod obiect absolut*, cu adrese fixate definitiv pentru instrucțiuni și date, sau *cod obiect relocabil*, în care se utilizează adrese temporare, fixarea definitivă a acestora fiind amânată pentru o etapă ulterioară. Această facilitate permite modularizarea programelor, care pot fi formate din mai multe fișiere sursă. Fiecare fișier sursă se assemblează separat, folosind un *asamblor relocabil*, după care modulele obiect generate la etapa anterioară sunt "legate" într-un singur fișier executabil, cu adrese fixe. În acest scop se utilizează un program de tip *editor de legături* (linker), care rezolvă referințele între modulele obiect și generează codul mașină pentru adresa de încărcare a acestuia în memorie.

Așa cum s-a menționat anterior, la scrierea programelor în limbaj de asamblare trebuie respectate anumite reguli sintactice, specifice limbajelor simbolice, definite de programul asamblor. Majoritatea asambloarelor acceptă fișiere sursă formate din linii de text ASCII având patru părți distincte, denumite *câmpuri*, și anume: *eticheta*, *cod*, *operand* și *comentariu*. Câmpurile

sunt despărțite prin *delimitatori de câmp*, care pot fi blankuri sau caractere speciale. Folosind notația BNF (Backus-Naur Form), o linie sursă în limbaj de asamblare se poate scrie:

<INSTR> ::= <ETICHETA>DC<COD>DC<OPERAND>DC<COMENTARIU>,

unde prin DC s-au specificat delimitatorii de câmp. Asamblorul trebuie să recunoască fiecare câmp și să-l interpreteze în mod corespunzător. Câmpurile COD și OPERAND corespund unei instrucțiuni din setul cu care este dotat microprocesorul sau unei pseudoinstrucțiuni specifice asamblorului.

Eticheta se folosește pentru a marca adresa instrucțiunii care urmează. Într-o instrucțiune nu este obligatorie prezența etichetei, aceasta fiind folosită în funcție de necesitățile programului. Obișnuit, o etichetă începe cu o literă iar lungimea sa maximă este o caracteristică a asamblorului. Eticheta poate fi definită astfel:

<ETICHETA> ::= <SIMBOL>

<SIMBOL> ::= <LITERA>|<LITERA><CHARACTER>|<LITERA><SIMBOL>

Codul este format numai din numele instrucțiunii sau al unei pseudoinstrucțiuni specifice limbajului de asamblare. De exemplu, pentru microprocesoarele Intel:

<COD> ::= MOV|ADC|PUSH|...|IN|OUT|HLT|ORG|...|DB

Operandul indică *adrese* sau *date*, necesare câmpului COD. În funcție de instrucțiune, operandul poate lipsi sau poate conține un număr finit de argumente, separate prin virgulă:

<OPERAND> ::= |<ARGUMENT>|<ARGUMENT>,...,<ARGUMENT>

În funcție de tipul asamblorului, există diferite moduri de exprimare a unui argument (nume de registre, constante numerice sau simbolice, etichete, expresii aritmetice sau logice):

**<ARGUMENT> ::= <CONSTANTA>|<SIMBOL>|
 <SIMBOL><OP><ARGUMENT>|
 <CONSTANTA><OP><ARGUMENT>**,

unde OP reprezintă un operator : **<OP> ::= + | - | * | /**.

Comentariul este alcătuit dintr-un șir de caractere alfanumerice și este utilizat de programator pentru descrierea operației executate de o instrucțiune sau de o secvență de instrucțiuni. Acest câmp este opțional.

Având în vedere și cele prezentate, pentru scrierea unui program în limbaj de asamblare trebuie cunoscute următoarele elemente caracteristice ale unui microprocesor:

- a) formatul instrucțiunilor;
- b) modurile de adresare;
- c) setul de instrucțiuni.

Formatul instrucțiunilor este specific fiecărui tip de microprocesor. Din punct de vedere formal, orice instrucțiune se împarte în câmpuri. La microprocesoare există numai două tipuri de câmpuri: câmpul *codului operației* sau *opcodul* (OPERation CODE) și câmpul care conține un *literal*. La sistemele de calcul clasice, câmpul codului operației se referă la codificarea operației și nu cuprinde nici o informație despre registrele interne. La microprocesoare, în acest câmp este inclus atât codul operației, precum și codul unor registre interne care vor fi utilizate în cursul execuției instrucțiunii. Rezultă un *cod al operației generalizat*, care prezintă avantajul creșterii vitezei de execuție și o dimensiune redusă a cuvântului mașină. Cel de-al doilea câmp, destinat păstrării unui literal, urmează întotdeauna după opcodul generalizat și conține o *dată* sau o *adresă*.

Modul de adresare reprezintă procedura prin care microprocesorul generează adresa locației unui operand care participă la execuția unei instrucțiuni. Modul de adresare influențează direct timpul necesar aducerii operandului în interiorul microprocesorului, deci și eficiența execuției. Fiecare tip de instrucțiune poate folosi unul sau mai multe moduri de adresare, aceasta fiind o măsură a capabilității și flexibilității unui microprocesor.

După cum s-a văzut mai sus, opcodul la microprocesoare poate conține și codul unor registre interne în care se află operandii. În fapt, acest mod de definire a adresei operandilor constituie un mod specific de adresare pentru microprocesoare, care poartă denumirea de **adresare la registru**. Dacă operandul se poate afla numai într-un registru special (de exemplu acumulatorul), atunci opcodul indică implicit adresa operandului - **adresare implicită**.

De asemenea, la microprocesoare se mai utilizează următoarele moduri de adresare:

- **adresarea imediată** - operandul se află în câmpul de literal al instrucțiunii, imediat după opcod;
- **adresarea directă** - adresa operandului se află în câmpul de literal al instrucțiunii;
- **adresarea indirectă** - câmpul de literal al instrucțiunii conține adresa locației de memorie unde se găsește adresa operandului.

În fig.1.7 sunt reprezentate schematic cele trei moduri de adresare mai sus menționate. La aceste moduri "naturale" de adresare se specifică adresa completă (absolută) a operandului, ceea ce conduce la o dimensiune mare a câmpului de adresă al instrucțiunii. Pentru a reduce lungimea cuvântului unui microprocesor este avantajoasă o definire *relativă* a adresei operandului. În acest caz, *adresa efectivă* (effective address) a operandului se obține prin însumarea conținutului câmpului literal al instrucțiunii, interpretat ca un număr binar în complement față de 2, cu conținutul unui *registru de bază*.

Dacă registrul de bază este contorul de program (PC) aceasta constituie **adresarea relativă**, când adresa efectivă a operandului se obține prin adunarea la conținutul PC a unui număr cu semn specificat în câmpul de literal al instrucțiunii. În cazul în care se folosește drept registru de bază un alt registru din interiorul microprocesorului se obține **adresarea indexată**, iar registrul de bază poartă denumirea de *registru index*.

În fig.1.7 sunt reprezentate schematic și modurile de adresare relativă și indexată. Pot exista și variante prin combinarea modurilor menționate sau prin includerea unei operații de incrementare/decrementare automată a registrului de bază. O variantă a adresării relative este **adresarea pe pagini**, în care adresa efectivă se obține prin simpla concatenare (fără adunare) a conținutului registrului de bază cu conținutul câmpului de literal al instrucțiunii.

Setul de instrucțiuni este de asemenea o caracteristică a unui tip de microprocesor. Fiecare microprocesor de uz general are un set specific de instrucțiuni, care constituie vocabularul

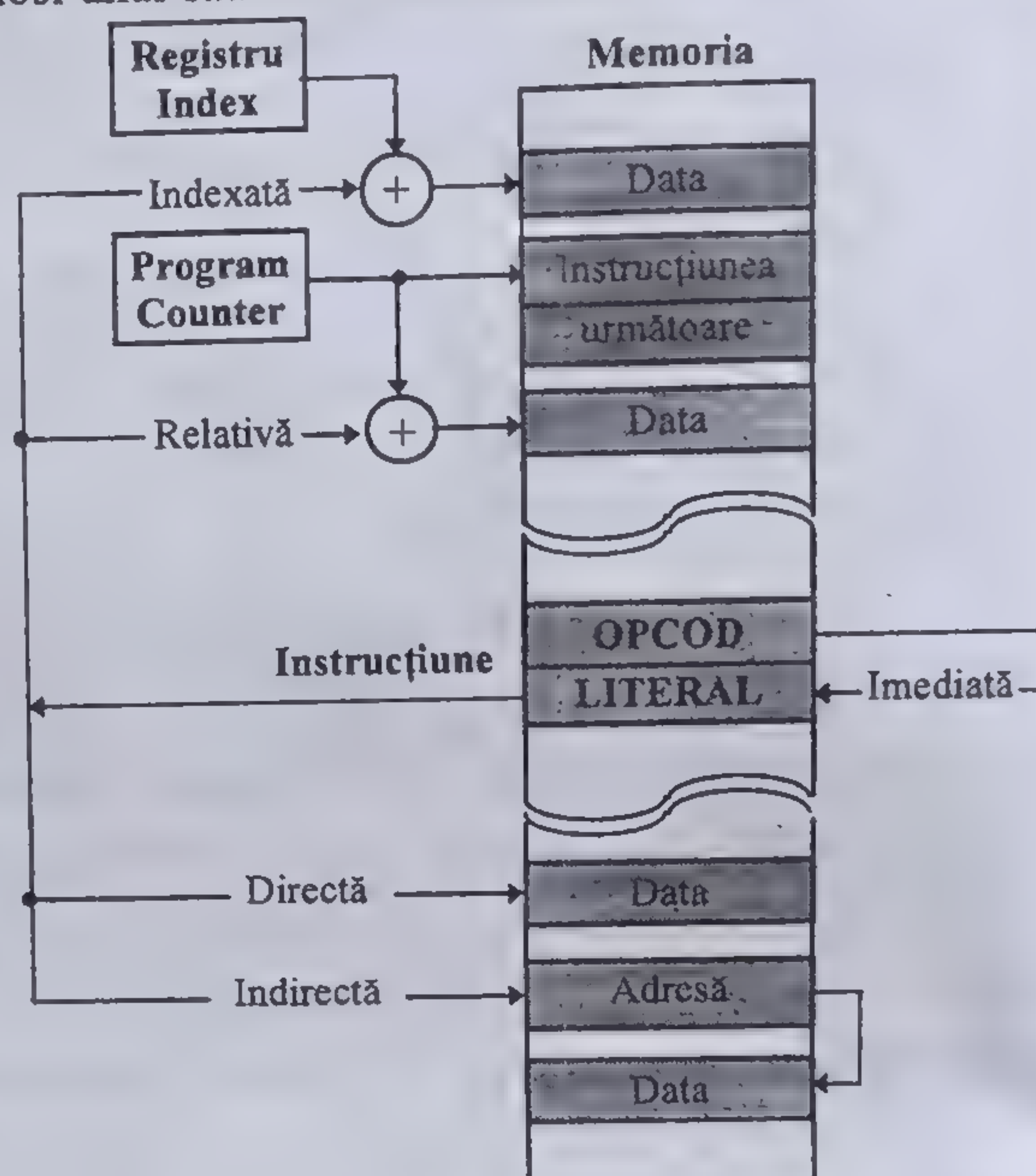


Fig.1.7. Reprezentarea principalelor moduri de adresare folosite la microprocesoare

limbajului său mașină. Setul de instrucțiuni este determinat de varietatea operațiilor pe care le poate executa un microprocesor. Deși fiecare producător are o clasificare proprie a instrucțiunilor, se poate considera că microprocesoarele sunt dotate cu următoarele tipuri de instrucțiuni:

a) **Instrucțiuni pentru realizarea operațiilor de bază.** În această categorie intră instrucțiuni pentru realizarea operațiilor logice și aritmetice.

b) **Instrucțiuni pentru operații asupra codurilor.** Cu această categorie de instrucțiuni se asigură operațiile de deplasare (logică, aritmetică sau ciclică) și de transfer al codurilor între memorie și registrele microprocesorului.

c) **Instrucțiuni pentru operații de comandă,** care asigură operațiile de ramificare (salt, apel și revenire - condiționate și necondiționate), de activare/dezactivare a întreruperilor sau de oprire a microprocesorului (HALT).

În cadrul programelor, după 4-5 operații de bază sau asupra codurilor urmează, în medie, una din operațiile de comandă. Aceste operații se utilizează atunci când este necesar să se modifice succesiunea naturală a execuției instrucțiunilor.

d) **Instrucțiuni pentru operații de I/E,** care asigură dialogul cu porturile sistemului în spațiul I/E. Durata operațiilor I/E este mult mai mare decât durata celorlalte categorii de operații. Aceasta se explică prin faptul că la efectuarea lor participă echipamente periferice sau interfețe de proces, cu convertoare A/D - D/A, care au o viteză mult mai mică decât microprocesorul. Din acest motiv se folosesc adesea interfețe de proces active (programabile), care lucrează relativ independent de sistem și dialoghează cu acesta prin întreruperi sau prin acces direct la memorie.

e) **Instrucțiuni pentru operații speciale,** care apar la microprocesoarele complexe, de 32 sau 64 biți și sunt folosite pentru controlul corectitudinii funcționării blocurilor sistemului, pentru managementul și protecția memoriei ș.a.

După cum s-a arătat, fiecare tip de microprocesor de uz general este dotat cu un set fix de instrucțiuni. Acest fapt poate reprezenta un dezavantaj major deoarece pentru unele aplicații sunt supraabundente și ineficiente. Pentru astfel de aplicații se poate apela la un tip special de microprocesoare, microprocesoarele *în felii* (bit slice), la care UC este microprogramabilă.

În acest caz, proiectantul de sistem poate implementa un set adecvat de instrucțiuni, dar numai dacă dispune de cunoștințe de microprogramare, fapt care a constituit o limitare puternică în dezvoltarea acestor sisteme.

În general, microprocesoarele conțin instrucțiuni din toate cele 5 grupe menționate mai sus, iar numărul lor total este între 50 și 150. Un număr mare de instrucțiuni are avantajul flexibilității, dar și dezavantajul unui mod mai greu de folosire. Folosind un număr mai mare de moduri de adresare, unii producători de microprocesoare au putut obține un set mai redus de instrucțiuni puternice. De exemplu, firma Motorola folosește la microprocesoarele sale de 8 biți: 7 moduri de adresare pentru 72 instrucțiuni la M6800 (prima generație), iar la M6809 (cel mai evoluat din generația a 2-a) 10 moduri de adresare la numai 59 de instrucțiuni. În acest fel, M6809 poate executa 1464 de variante de instrucțiuni față de numai 195 la M6800 [34]. Realizarea programelor în limbaj de asamblare prezintă avantajul unei eficiențe ridicate, deoarece permite manipularea individuală a registrelor și a unor biți de condiție ai microprocesoarelor. Drept urmare, rezultă programe compacte și care generează coduri obiect de dimensiuni reduse, lucru foarte important în aplicațiile de timp real.

2) **Limbajele de nivel înalt** elimină dificultățile impuse de limbajul de asamblare privitor la necesitatea cunoașterii în detaliu de către programator a resurselor interne ale microprocesorului. Aceste limbaje sunt mai apropiate de limbajul natural decât de limbajul mașină, deoarece urmăresc în mai mare măsură exprimarea algoritmică a problemelor. Ca limbaje de nivel înalt mai des utilizate se pot menționa: C, PL/M (dezvoltat de Intel), BASIC, FORTH, PASCAL. Limbajele de nivel înalt sunt eficiente în cazul scrierii programelor complexe, unei instrucțiuni în limbaj înalt

corespunzându-i un grup de instrucțiuni în limbaj mașină. Astfel, prin efectuarea operației de *compilare*, se generează un cod obiect relativ voluminos și lent. Acest dezavantaj poate fi redus parțial prin utilizarea unor compilatoare performante, care minimizează dimensiunea și cresc viteza de execuție a codului obiect generat. Aceste limbaje nu formează obiectul lucrării de față, de aceea nu se va mai insista asupra lor.

Etapele realizării unui program

La realizarea unui program se parcurg mai multe etape succesive, principale fiind cele prezentate în fig.1.8. Analiza problemei permite stabilirea principalelor fluxuri informaționale și reprezentarea lor cu ajutorul *organigramelor*. O astfel de reprezentare a problemei ajută mult la precizarea tuturor detaliilor, fiind în același timp utilă și la depanare. În plus, problema reprezentată prin organigramă permite o mai ușoară *proiectare modulară* a programului. Avantajele realizării unui program din mai module decurg din verificarea lor facilă și din simplificarea scrierii acestuia.

După asamblare/compilare urmează faza de editare a legăturilor, apoi programul executabil rezultat se încarcă în memoria sistemului, se execută și se verifică sub controlul unui *program monitor*. De obicei sunt necesare mai multe iterații până la obținerea rezultatului dorit. Programul monitor este un nucleu care asigură inițializarea și controlul resurselor sistemului cu microprocesor și care facilitează încărcarea programului în memorie și execuția acestuia. Mai multe detalii referitoare la etapele care se parcurg la realizarea unui program sunt prezentate în [4].

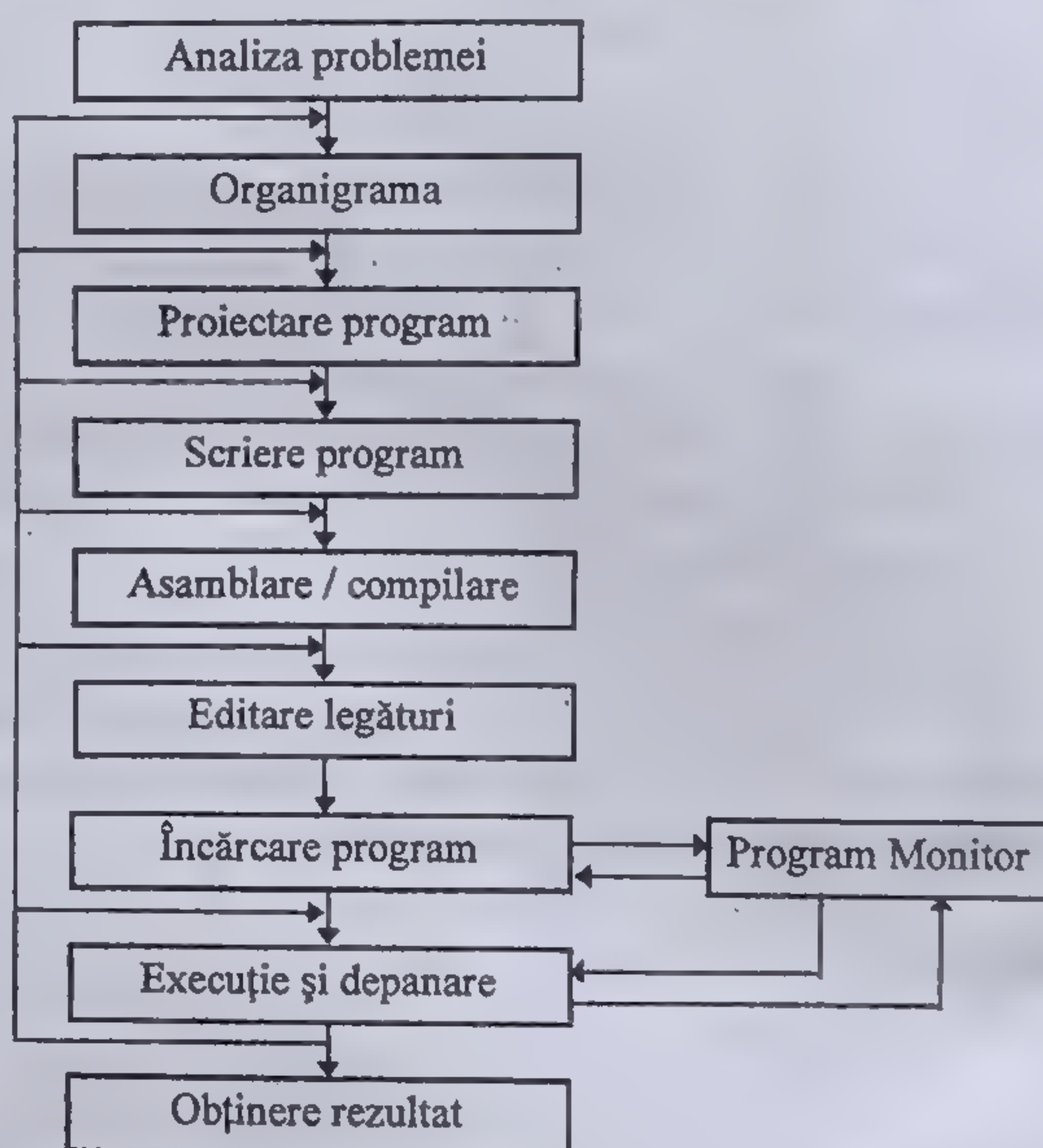


Fig.1.8. Principalele etape care se parcurg la realizarea unui program

1.1.1.5. Evoluția microprocesoarelor. Tipuri principale

Microprocesoare de uz general

Primul microprocesor, cu indicativul I4004, a fost realizat de firma Intel în 1971, avea lungimea cuvântului de 4 biți și conținea aproximativ 2300 tranzistoare. Succesul neașteptat raportat de acest dispozitiv în rândul proiectanților de structuri numerice, prin programabilitatea sa, a fost factorul decisiv în evoluția microprocesoarelor. Astfel, în 1972 tot Intel lansează I8008, capabil să lucreze cu cuvinte de 8 biți și care reprezintă *prima generație* de microprocesoare. Unii utilizatori s-au orientat imediat către realizarea de microcalculatoare. Dar, un 8008 necesita mai mult de 20 de dispozitive adiționale pentru a forma o unitate centrală de procesare.

În 1974 Intel anunță microprocesorul I8080, care are un set de instrucțiuni mai elevat decât 8008 și necesită numai două circuite adiționale pentru a forma o UCP funcțională. De asemenea, 8080 este realizat în tehnologie NMOS, care-i asigură o viteză de operare mult mai ridicată decât a predecesorului său. Acest microprocesor deschide seria celei de-a doua generații de microprocesoare și devine referință pentru familia de 8 biți.

După un timp relativ scurt de la lansarea lui 8080, în competiție intră firma Motorola cu produsul M6800. Microprocesorul 6800 are avantajul că necesită o singură sursă de alimentare de +5V față de cele trei (-5V, +5V și +12V) necesare la 8080. Apoi, Intel produce în 1976 microprocesorul 8085, o variantă îmbunătățită a lui 8080. Imediat Motorola produce M6809, care are multe instrucțiuni pentru 16 biți.

Deși mult timp I8080 și M6800 au deținut topul vânzărilor, în competiție au intrat și alți producători importanți precum RCA (cu CDP1802, în tehnologie CMOS), Fairchild (cu F8), National Semiconductors sau Signetics, dar și MOS Technology cu 6502 (folosit ca UCP de Apple), respectiv Zilog cu microprocesorul Z80.

Tot firma Intel lansează în 1978 cea de-a *treia generație*, a microprocesoarelor de 16 biți, prin produsul I8086 considerat "standard" pentru generația sa, precum a fost considerat 8080 pentru cele de 8 biți. Deși firmele National Semiconductors, cu produsul PACE, și Texas Instruments, cu TI9900 (încă din 1977) sunt primele care au realizat microprocesorul de 16 biți, acestea nu au adus noutăți majore în concepție și au fost repede abandonate. La scurt timp după apariția lui 8086, firma Motorola lansează M68000 iar Zilog - Z8000.

Aceste microprocesoare nu numai că lucrează direct cu cuvinte de 16 biți și pot adresa mai mult de 1 Moctet de memorie, față de numai 64 Kocteți la cele de 8 biți, dar și execută mult mai rapid instrucțiunile. De asemenea, au instrucțiuni pentru operații de înmulțire și împărțire a numerelor întregi, care nu se regăsesc la cele de 8 biți, fapt ce le face mai atractive pentru implementarea algoritmilor de conducere automată.

Microprocesoare pentru unitățile centrale ale microcalculatoarelor

Consacrarea produselor Intel pe piața microcalculatoarelor începe cu anul 1981, când IBM decide să producă primele calculatoare personale IBM PC-XT cu microprocesoarele 8086/8088. I8088 conține în interior un 8086 dar are o magistrală externă de date de 8 biți pentru a beneficia de familia numeroasă de componente LSI dezvoltată în jurul lui 8080. Urmează I80186/80188 care deschide linia de integrare pe același cip a unor porturi și timere. Urmează cel mai performant microprocesor de 16 biți al firmei Intel - 80286, care prefigurează prin noile concepte introduse o nouă *generație*, cea de-a *patra*. Pe baza acestui produs și IBM lansează o nouă generație de calculatoare personale: IBM PC-AT (Advanced Technology).

Proiectanții de sistem au găsit din ce în ce mai multe aplicații pentru microprocesoare, chiar de la primele generații de 8 biți. Aceștia au exercitat o presiune asupra producătorilor de microprocesoare pentru realizarea de dispozitive cu arhitecturi și facilități optimizate pentru anumite tipuri de aplicații. Tendința a fost către realizarea de unități centrale pentru microcalculatoare. Dar, abia începând cu cea de-a *patra* generație de microprocesoare, orientarea s-a făcut major în acest sens.

Deși această orientare a fost vizibilă chiar de la începutul realizării microprocesoarelor, impactul major s-a produs efectiv odată cu realizarea în 1986 a primului microprocesor de 32 de biți de către Intel (i386). Progresele obținute în tehnologia de integrare pe corp solid au permis proiectarea unor UCP pe un singur cip SVLSI, cu facilități specifice minicalculatoarelor clasice. Astfel de procesoare pot adresa memorii de ordinul giga- (10^9) sau tera- (10^{12}) octeților și conțin cele mai noi concepte arhitecturale din domeniul tehnicii de calcul [3].

Microprocesorul i386 aduce îmbunătățiri fundamentale predecesorului său (80286), dar a reprezentat și o performanță tehnologică a momentului: 275 de mii de tranzistoare, tehnologie CMOS cu trasee de 1,5 μm . Este urmat de M68020 al firmei Motorola și de NS32032 al firmei National Semiconductors. În anul 1989 (cu un an mai devreme decât era prevăzut) se produce i486, care depășește 1 milion de tranzistoare echivalente (1,2 milioane) și funcționează cu frecvențe de tact între 33 și 100 MHz. Aceste microprocesoare permit o evoluție nebănuită în

domeniul microcalculatoarelor personale, cu performanțe ce depășesc cu mult pe cele ale minicalculatoarelor clasice.

Deși eforturile materiale și de cercetare pentru dezvoltarea de noi produse devin impresionante, ritmul de înnoire nu scade, ci dimpotrivă. Astfel, Intel lansează în 1993 primul microprocesor al generației a cincea, numit din acest motiv chiar **Pentium (i586)**. Acest super-microprocesor inaugurează seria de procesoare cu magistrala de date de 64 de biți, cu o arhitectură puternic diferită de a predecesoarelor și care, printr-o tehnologie BICMOS, folosind trasee de numai 0,8 μm , înglobează 3,1 mil. de tranzistoare echivalente. Au urmat, într-un ritm de inovare fantastic, Pentium MMX, Pentium II (300MH - apărut în 1997), Pentium III (1,15 GHz - 1999) și Pentium 4 (2 GHz - 2000/01) pe 128 biți și cu peste 40 milioane de tranzistoare [www.intel.com].

Cele prezentate mai sus se referă exclusiv la microprocesoarele ale căror caracteristici principale s-au dezvoltat plecând de la tipurile de bază din anii '70. Astfel de microprocesoare pot fi denumite *de uz general* și s-au dezvoltat în *conceptul de arhitectură CISC* (Complex Instruction Set Computer). Trebuie menționat faptul că există și o altă direcție în care au evoluat procesoarele pentru tehnica de calcul, așa numitele *procesoare RISC* (Reduced Instruction Set Computer). Printre acestea se pot menționa microprocesoarele i960 (Intel) și M88000 (Motorola).

Microcontrolere

O altă direcție de dezvoltare, care în prezent a căpătat amploare, este cea a structurilor cu microprocesoare "împachetate" (embedded microprocessors). Astfel de dispozitive au apărut din necesitatea punerii la dispoziția proiectanților de echipamente "inteligente" (de la periferice pentru calculatoare la bunuri de larg consum) a unor structuri integrate pe un singur cip, având resursele unui sistem complet cu microprocesor. Evident, o astfel de dorință a condus la schimbarea opticii, atât a proiectanților de microcipuri, cât și a proiectanților de sisteme. Astfel, producătorii de controlere "împachetate" au căutat să includă pe același cip în afară de un microprocesor clasic și memorie, porturi I/E, timere, controlere de întreruperi sau convertoare A/D și uneori D/A. Limitarea numărului de pini ai capsulei a impus realizarea nu a unor produse singulare, ci a unor familii numeroase, grupate în jurul unui tip de bază. Această orientare în realizarea microprocesoarelor este extrem de benefică și pentru aplicațiile de control automat, în esență pe un cip fiind conținute elementele esențiale ale unui microsistem pentru aplicații de timp real - de unde și denumirea actuală, devenită consacrată, de *microcontroler*. Evoluția tehnologică a microcontrolerelor a permis orientarea lor spre realizarea *conducerii distribuite*, prin includerea pe cip și a unor circuite specializate de comunicație serială. Firma Intel a introdus chiar un standard nou de comunicație la nivel de bit - BITBUS™ - și a generat pachete de programe de dezvoltare a aplicațiilor, care să vină în ajutorul proiectanților de sistem.

Unul dintre primii producători de microcontrolere este Texas Instruments. Prin produsul TMS-1000, TI oferă o familie variată de microprocesoare de 4 biți, deosebit de apreciate și folosite și în prezent în realizarea de "smart machines" (mașini de spălat automate, cuptoare cu microunde programabile etc.). În anul 1976, Intel produce primul său *microcalculator integrat pe un cip* - MCS-48 - care conține un microprocesor 8080, RAM, ROM și 3 porturi I/E, toate într-o capsulă cu 40 de pini. În prezent această firmă produce familia de microcontrolere de 8 biți MCS-51, iar Motorola - familia MC68HC11, ambele deosebit de apreciate. Dar și în acest domeniu evoluția este extrem de rapidă, fiind produse microcontrolere foarte performante de 16 biți (de exemplu familia MCS-96 a firmei Intel sau familia MC68HC16 a firmei Motorola) și de 32 de biți: i80960 (i960) - Intel, Am29050 - AMD sau NS32SF641 - National Semiconductor [15]. Ultimele produse, de 32 de biți, includ în arhitectură unele dintre cele mai evaluate concepte din domeniul tehnicii de calcul. De exemplu, i960 este un procesor superscalar care combină cele mai bune caracteristici ale sistemului CISC cu implementarea eficientă a proiectelor RISC și a fost conceput să poată fi folosit

Capitolul 1 - Structura sistemelor cu microprocesoare

Într-o gamă largă de implementări tehnologice, precum: robotică, aviație și aerospațiu, dar și în birotică sau instrumentație medicală [15].

Microprocesoare specializate

Concepute pentru a fi utilizate într-o gamă largă de aplicații, microprocesoarele de uz general nu pot realiza eficient sarcini speciale, fapt ce a condus la realizarea de procesoare dedicate. Printre acestea se numără *procesoarele matematice*, specializate pentru calculul HW a funcțiilor aritmetice de uz general: operații în virgulă flotantă, funcții trigonometrice și exponențiale, radicali de ordinul 2, logaritmi zecimali și naturali etc. Astfel de procesoare au fost concepute să funcționeze în paralel cu microprocesoarele de uz general și să poată comunica cu ele. Din acest motiv mai sunt cunoscute sub denumirea de *coprocesoare matematice*. Includerea lor în sistem mărește considerabil eficiența ansamblului, în cazul unor aplicații ce reclamă calcule complexe și este deosebit de avantajoasă la conducerea unor procese rapide, cum ar fi cele din domenii ca: acționări electrice, comanda numerică a mașinilor unelte, energetică ș.a.

Un astfel de coprocesor este produsul Intel 8231, care poate fi interfațat cu orice tip de microprocesor printr-o magistrală de date de 8 biți. Pentru a mări eficiența la microcalculatoarele de uz general, firma Intel produce coprocesoare matematice direct compatibile cu microprocesoarele care formează UCP: 8087 pentru 8086, 80287 pentru 80286, 80387 pentru i386. Următoarele variante (i486, Pentium) includ pe cip și unități de calcul în virgulă mobilă.

O altă categorie de microprocesoare specializate o formează *procesoarele numerice de semnal* (Digital Signal Processors), orientate spre implementarea algoritmilor specifici prelucrării semnalelor: transformata Fourier rapidă (FFT), filtre digitale, produse de corelație etc. Și aceste procesoare sunt deosebit de atractive pentru aplicații specifice domeniului automaticii. Dintre cele mai cunoscute, putem menționa familia de DSP-uri ale firmei Texas Instruments - TMS320Cx ($x=1\div 8$) sau pe cele ale firmei Analog Devices ADSP2100 și ADSP21000.

Microprocesoare "în felii" (Bit-Slice Processors)

Microprocesoarelor discutate până acum sunt formate dintr-o singură capsulă și au un set fix de instrucțiuni. Dar, pentru unele aplicații, microprocesoarele de uz general nu sunt suficient de rapide sau setul de instrucțiuni nu este cel mai potrivit. Pentru astfel de aplicații s-au dezvoltat microprocesoare "în felii", care permit organizarea cea mai convenabilă a UCP, precum și a setului de instrucțiuni. Exemple de astfel de microprocesoare sunt cele produse de Intel - seria 3000 - și AMD 2900 al firmei Advanced Micro Devices. Astfel, familia 2900 include ULA de 4 biți, multiplexoare, secvențiatoare și alte elemente necesare construirii unei UCP. Cu ajutorul acestora, prin conectarea lor în paralel, se pot construi unități centrale cu lungimea dorită a cuvântului procesat - 8, 16 sau 32 biți, în funcție de necesitățile unui anumit tip de aplicație. De la această posibilitate de conectare - în paralel, pe "felii de biți" - a rezultat și denumirea clasei de microprocesoare.

În acest caz, proiectantul nu concepe numai structura HW, ci și setul de instrucțiuni - prin microprogramare la nivel de microcod [28]. Se poate obține astfel un sistem de calcul de mare viteză, adaptat la maximum la cerințele aplicației. Evident, acest fapt atrage după sine o pierdere a versatilității și în plus necesitatea ca proiectantul să posede cunoștințe solide de microprogramare.

Asemenea tehnici se utilizează la realizarea procesoarelor de semnal, precum și la realizarea de calcule aritmetice complexe, cu aplicații directe în comanda numerică a mașinilor unelte, recunoașterea formelor etc. Apariția procesoarelor "single-chip" de tip DSP [41] sau a coprocesoarelor matematice a diminuat oarecum dezvoltarea procesoarelor "bit-slice". Astfel, Texas Instruments, prin familia TMS320, este lider mondial în domeniul DSP-urilor, cu peste 50% din piață. Urmează Motorola cu seria 560xx și Analog Devices cu seriile 21xx (16 biți) și 2100x (32 biți). În prezent există tendința de a realiza pe același cip atât DSP-ul cât și microcontrolerul, ceea ce aduce avantaje deosebite de performanță și viteză, reducând totodată efortul proiectantului.

1.1.2. Memoria sistemelor cu microprocesoare

Această componentă a sistemului asigură păstrarea instrucțiunilor programului și a datelor. Din punct de vedere fizic, blocul de memorie (MEM) se compune din două tipuri de elemente (fig.1.9):

- **memoria de program (MEMP)**, care este o memorie permanentă (nevolatilă), din care se pot efectua numai citiri. În MEMP se stochează secvențele de instrucțiuni (programe) și constantele programului. În sistemele cu microprocesoare această parte a memoriei se realizează cu dispozitive integrate de tip ROM (Read Only Memory) și PROM (Programmable ROM);

- **memoria de date (MEMD)**, care permite păstrarea temporară a datelor variabile și poate fi atât citită, cât și înscrisă; se realizează fizic cu circuite integrate de tip RAM (Random Access Memory), statice (SRAM) sau dinamice (DRAM).

Din punct de vedere conceptual, împărțirea în MEMP și MEMD nu este necesară. Din considerente practice, este util să existe o *memorie nucleu* care să păstreze programele necesare funcționării corecte a sistemului și al cărei conținut să nu se piardă odată cu dispariția tensiunii de alimentare.

Memoria sistemului se organizează liniar, de obicei în cuvinte cu dimensiunea identică cu a cuvântului procesorului, deci $l_{\text{CMEM}} = d$, unde l_{CMEM} reprezintă lungimea cuvântului stocat în memorie. Ca urmare, blocul MEM poate fi privit ca un spațiu M , denumit *spațiul memoriei*, în care se pot memora cuvinte binare de lungime d . Numărul maxim de cuvinte al spațiului M care pot fi adresate *direct* de către microprocesor este determinat de dimensiunea a a magistralei de adrese. Rezultă deci că spațiul adreselor, A , conține 2^a adrese cu care pot fi selectate cuvinte din spațiul memoriei.

Așadar, se poate spune că *operația de adresare* alocă unei adrese din A un element din M prin funcția de traducere :

$$f_T: A \rightarrow M$$

Dacă $A=M$, atunci f_T este *funcția identitate* (fig.1.10). La multe sisteme pentru conducere automată dimensiunea memoriei, M , este mai redusă decât capacitatea A de adresare a microprocesorului.

Dacă dimensiunea MEM depășește spațiul de adresare directă a microprocesorului, cum este la sistemele de calcul, atunci se poate folosi o *memorie auxiliară* (MAUX), dispusă în afara *memoriei principale* formată din MEMP+MEMD. În acest caz, MEMP conține un nucleu care permite încărcarea în MEMD a informațiilor din MAUX, realizată pe suporturi externe mult mai lente decât memoriile semiconductoare (discuri flexibile sau fixe, CD, bandă magnetică).

Important pentru sistemele cu microprocesoare este modul de realizare a memoriei principale: MEMP+MEMD. După cum s-a arătat, aceasta este de

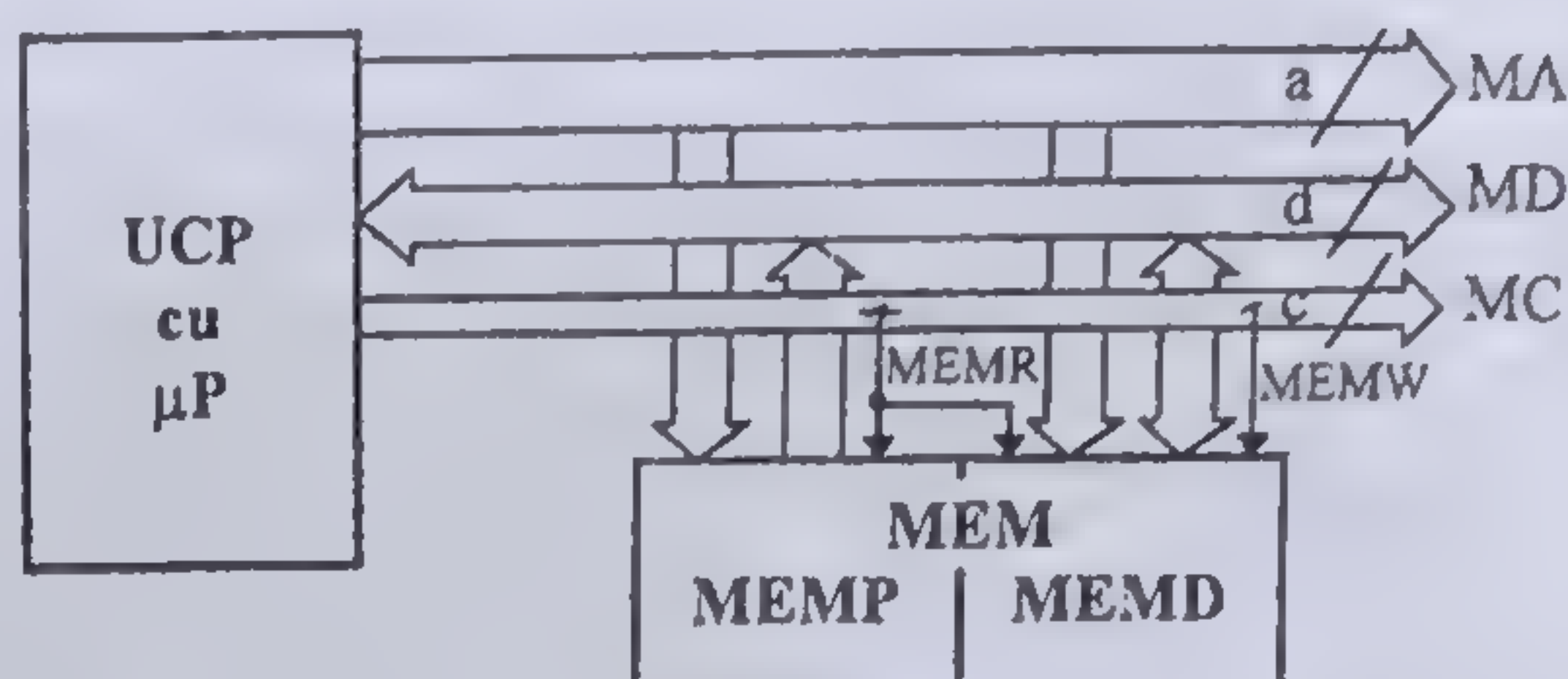


Fig.1.9. Structura blocului de memorie

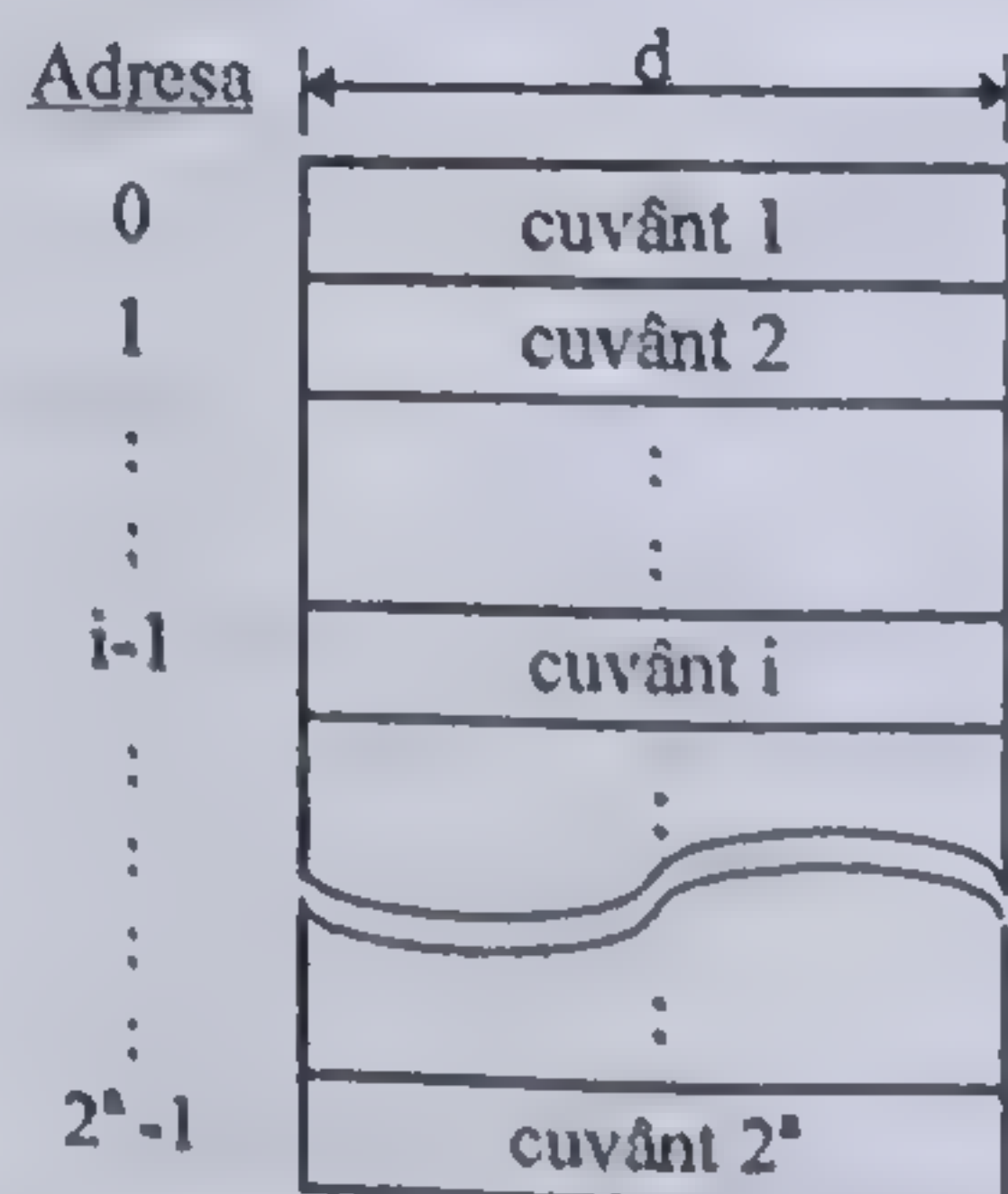


Fig.1.10. Organizarea memoriei

obicei realizată cu dispozitive semiconductoare. Pentru o corectă funcționare a sistemului, proiectantul trebuie să cunoască modul în care se desfășoară transferul de informație între MEM și microprocesor, pentru a putea conecta în mod adecvat dispozitivele de memorie la magistrala sistemului.

1.1.2.1. Transferul de informație între memorie și microprocesor

Pentru realizarea unui transfer de informație, microprocesorul trebuie să furnizeze semnalele de comandă necesare pentru cele două cicluri fundamentale:

- de înscrisere a datelor în memorie;
- de citire a datelor din memorie.

Pentru o corectă înțelegere a operațiunilor pe care le execută un microprocesor în dialogul cu circuitele de memorie este necesar să se cunoască condițiile cerute de aceste dispozitive.

Scrierea datelor în memorie

Considerăm o memorie cu n linii de adrese, m linii de date și o linie de comandă a scrierii, notată \overline{WR} (Write). În fig.1.11 este prezentată evoluția semnalelor, necesară în cazul operației de înscrisere a unei date. Ca atare, trebuie asigurată următoarea succesiune temporală de operații:

- se aplică adresa pe liniile $A_n \div A_1$;
- se aplică datele pe liniile $D_m \div D_1$;
- după timpul t_{DS} (Data Set), necesar stabilizării datei, se aplică semnalul de comandă \overline{WR} , de durată T_{WP} (Write Pulse);
- după aplicarea comenzii \overline{WR} , datele se mențin neschimbate pe liniile $D_m \div D_1$ un timp t_{DH} (Data Hold).

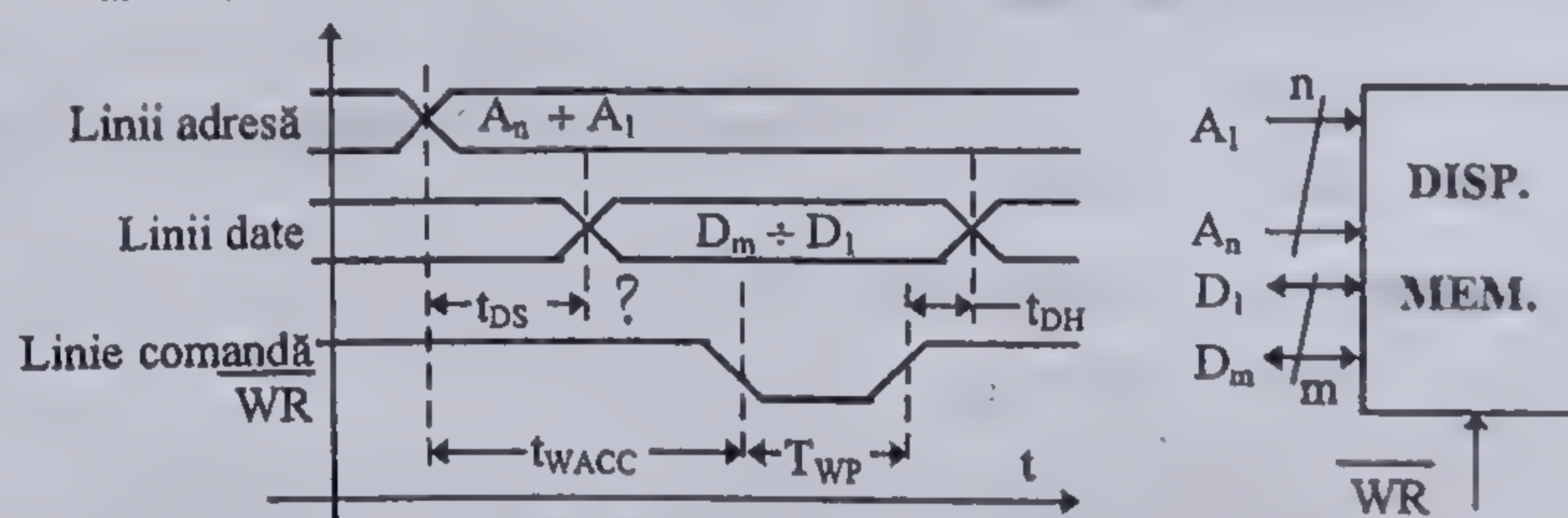


Fig.1.11. Succesiunea semnalelor la scrierea într-un dispozitiv de memorie

Caracteristic pentru un dispozitiv de memorie este intervalul de timp dintre momentul în care s-a aplicat adresa și momentul în care se poate aplica impulsul de scriere, denumit *timp de acces al scrierii* - t_{WACC} (Write Access time). Cunoașterea acestei caracteristici este esențială pentru o bună proiectare a sistemului. În funcție de tehnologia de realizare, t_{WACC} se situează la cca. 200 ns pentru memoriile MOS și poate coborâ până la cca. 30 ns pentru cele bipolare (TTL).

Secvențierea în timp a semnalelor ce trebuie aplicate unui dispozitiv de memorie la înscrisere definește microoperațiile care trebuie realizate pentru o procedură de înscrisere. Considerând că informația se află în registrul R al microprocesorului și urmează să fie transferată în MEM la adresa α , transferul implică următoarele două faze:

a) pregătirea memoriei

$$\left. \begin{array}{l} MA \leftarrow \alpha \\ MD \leftarrow (R) \\ MC \leftarrow \overline{MEMW} \end{array} \right\} \text{transmise succesiv de către microprocesor}$$

b) înscriserea informației

$$MEM(\alpha) \leftarrow (MD)$$

În secvențele de mai sus, prin (.) s-a notat conținutul unui registru sau al unei magistrale, iar prin $\text{MEM}(\alpha)$ - locația de memorie de adresă α . Comanda $\overline{\text{MEMW}}$ este transmisă de către microprocesor pe linia $\overline{\text{WR}}$ a memoriei.

Citirea datelor din memorie

Diagramele tipice în acest caz sunt cele din fig.1.12 și exprimă următoarea secvență:

- se depune adresa pe liniile $A_n \div A_1$;

- se comandă citirea datei, după un timp t_{RACC} , când memoria poate depune data pe liniile $D_m \div D_1$.

Rezultă următoarea descriere sintetică a operațiilor în cazul unei proceduri de citire a informației de la adresa α din MEM în registrul R al microprocesorului:

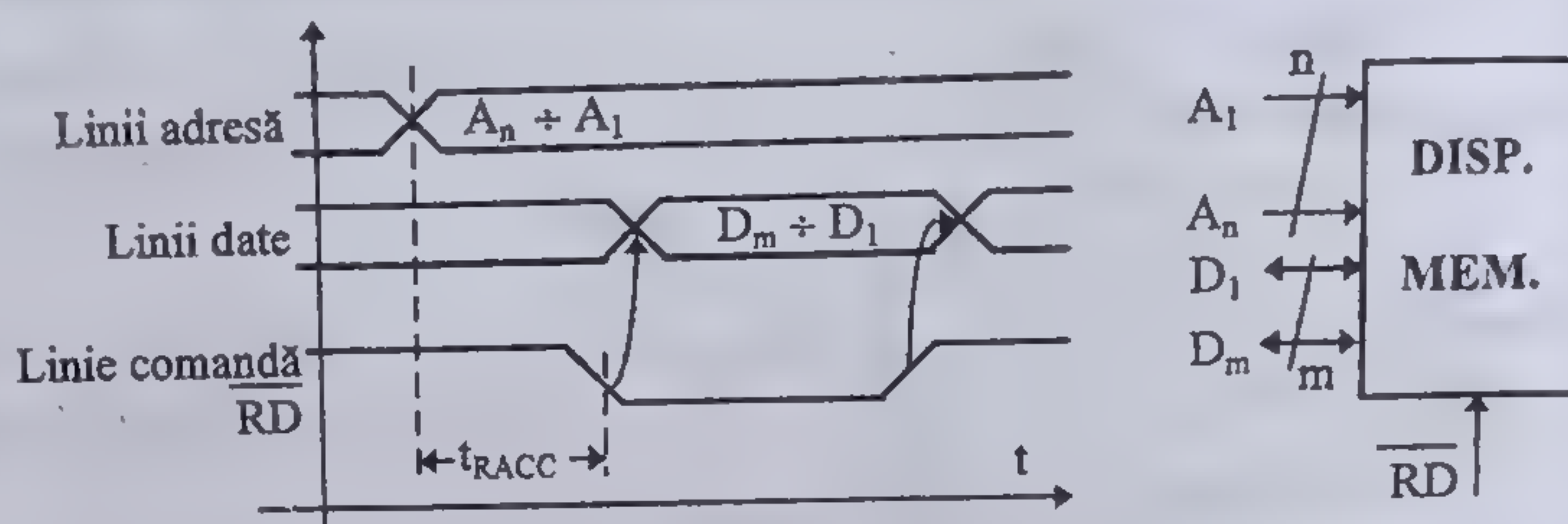


Fig.1.12. Succesiunea semnalelor la citirea dintr-un dispozitiv de memorie

a) pregătirea memoriei

$\text{MA} \leftarrow \alpha$
 $\text{MC} \leftarrow \overline{\text{MEMR}}$

} aplicate succesiv de către microprocesor

b) citirea propriu-zisă

$\text{MD} \leftarrow \text{MEM}(\alpha)$
 $\text{R} \leftarrow (\text{MD})$

Comanda $\overline{\text{MEMR}}$ se transmite de către microprocesor pe linia $\overline{\text{RD}}$ a memoriei. Multe dispozitive de tip RAM folosesc o singură linie de comandă, de tip $\text{R}/\overline{\text{W}}$, când ambele valori ale semnalului de comandă sunt semnificative. Acest fapt simplifică controlul realizat de microprocesor, care trebuie să transmită numai semnalul $\overline{\text{MEMW}}$ la memoria RAM.

De obicei, timpul de acces (t_{ACC}) este mai mare la citire ($t_{\text{RACC}} > t_{\text{WACC}}$) și din acest motiv se ia în considerație la proiectarea memoriei sistemului. Pentru ca un microprocesor să poată efectua un transfer corect cu memoria este necesar ca $t_{\text{ACC}} \leq t_{\text{CY}}$. Dacă nu se poate îndeplini această condiție, proiectantul trebuie să prevadă o logică de așteptare.

1.1.2.2. Conectarea memorilor statice la magistrala sistemului

În afara condiției de asigurare a timpului de acces, pentru o corectă proiectare a memoriei este necesar să se determine numărul dispozitivelor de memorie și modul lor de conectare la sistem. Ultimul aspect este impus de faptul că producătorii de memorii semiconductoare construiesc un număr limitat de tipuri, cu care un proiectant trebuie să asigure necesarul de memorie cerut de aplicație. Criteriul economic impune folosirea celui mai redus număr de dispozitive, care să fie complet utilizate. Din cele prezentate în acest capitol, rezultă că trebuie rezolvate următoarele trei aspecte:

- asigurarea condiției $d = l_{\text{CMEM}}$; în cazul în care $d > l_{\text{CMEM}}$, trebuie să se conecteze mai multe dispozitive în paralel;
- stabilirea numărului de dispozitive pentru a satisface necesarul de MEMP și MEMD;
- selecția corespunzătoare a dispozitivelor de memorie.

Capitolul 1 - Structura sistemelor cu microprocesoare

Exemplul 1.1. Necesarul de memorie de date pentru un sistem cu $d = 8$ este de 4Ko. Se dispune de dispozitive de memorie RAM de 1Ko și $l_c = 4$.

a) Pentru că $l_{\text{CMEM}} = d = 8$, trebuie conectate câte două dispozitive în paralel la magistrala de date pentru fiecare 1Ko, ca în figura 1.13a. De asemenea, este necesar ca ambele dispozitive să fie controlate simultan de către microprocesor, prin liniile R/W.

b) Realizarea necesarului de 4Ko impune folosirea a 4 grupuri (bancuri) de 1Ko fiecare (fig.1.13b), deci în total 8 dispozitive de $1K \times 4$ biți. Cele 4 grupuri se vor conecta la magistrala de date ca în schema din fig.1.13a.

c) Întrucât aceste grupuri sunt conectate identic la magistrală, dar trebuie să aibă adrese distincte, pentru asigurarea spațiului de 4Ko apare necesitatea *selectării* individuale, de către microprocesor, a celor 4 bancuri. Pentru selecție trebuie ca fiecare dispozitiv de memorie să fie prevăzut cu un pin de selecție.

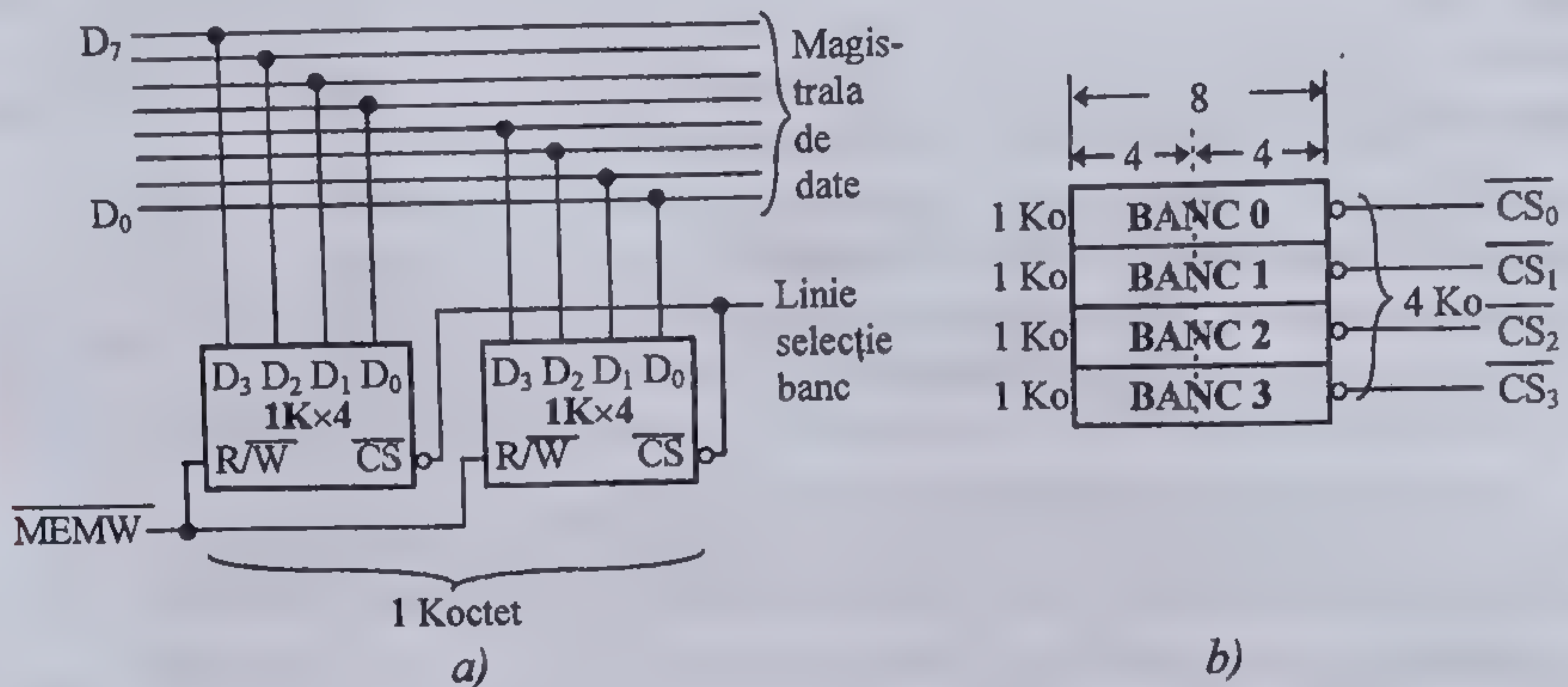


Fig.1.13. Realizarea unei memorii de 4Ko cu dispozitive de $1K \times 4$ biți

Selectarea dispozitivelor de memorie se poate face prin intermediul liniilor de adresă. În principiu, există trei modalități de selecție:

- **selecția liniară**, când fiecărei memorii sau grup de memorii care formează un banc i se atribuie o linie de adresă;
- **selecția cu decodificare**, când între liniile de adresă și bancurile de memorie se folosește un nivel de decodificare/demultiplexare;
- **selecția combinată**, când se utilizează simultan primele două soluții.

Prima modalitate este avantajoasă la realizarea sistemelor mici, iar a doua în cazul sistemelor dezvoltate, întrucât asigură o folosire economică a magistralei de adrese.

Proiectarea memoriei statice a unui sistem cu microprocesor

Pentru a realiza o proiectare sistematică, ținând cont de cele prezentate mai sus, este necesar să se parcurgă următoarele etape:

- 1) În corespondență cu dimensiunea MEMP, se aleg dispozitivele PROM care să asigure $t_{\text{ACC}} \leq t_{\text{CY}}$ și, dacă este posibil, numărul cel mai redus de circuite.
- 2) Similar pentru MEMD, folosind dispozitive RAM.
- 3) Se organizează *harta memoriei*, cu marcarea spațiului ocupat de fiecare dispozitiv/grup de dispozitive. Obişnuit, de la adresa de debut obținută după inițializarea microprocesorului se dispune MEMP și apoi MEMD.
- 4) Se stabilește *tabelul de adresare și selecție*, care conține modul de utilizare a liniilor magistralei de adrese, pentru a evita un conflict logic. Pentru o încărcare echilibrată a MA este

avantajos ca o parte din linii să fie folosite pentru *adresarea* locațiilor de memorie, iar o parte din liniile rămase pentru *selectarea* bancurilor de memorie.

5) Se realizează schema de conectare a dispozitivelor de memorie la magistrala sistemului, în corespondență cu tipul de selecție și particularitățile de control ale dispozitivelor.

Exemplul 1.2. Se consideră un sistem cu microprocesor de 8 biți, având $t_{CY} = 500$ ns și $\alpha=16$ ($A = 64Ko$), care necesită o memorie operativă formată din 12Ko memorie program și 2Ko memorie de date. Se dispune de memorii EPROM de 4Ko (2732) și RAM de 2 Ko (6116) și care asigură timpul de acces ($t_{ACC} < 330$ ns).

1÷2) Sunt necesare 3 dispozitive EPROM și 1 dispozitiv RAM.

3) Considerând că adresa de debut este 0000h se obține harta memoriei din figura 1.14. Memoria program ocupă spațiul dintre adresele 0000h ÷ 2FFFh, iar memoria de date spațiul de 2Ko începând de la adresa 3000h. În acest caz s-a realizat o ocupare compactă a spațiului de memorie, spațiul începând de la adresa 3800h fiind liber. Acest mod contiguu de aranjare a zonelor de memorie simplifică și logica de selecție a dispozitivelor. Restul spațiului poate fi folosit pentru adresarea porturilor I/E sau pentru extensia sistemului.

4) Tabelul de adresare și selecție trebuie să conțină modul de utilizare a liniilor de adresă internă, pentru selecția locațiilor de memorie și respectiv a liniilor de selecție pentru activarea celor 4 dispozitive de memorie (tab.1.1).

Prin "x" s-a notat valoarea indiferentă (0 sau 1) pe care o pot lua biții de adresă. În tabel se specifică folosirea liniilor de adresă în cazul selecției liniare și pentru selecția cu decodificare. În ultimul caz, numărul de linii pentru selecție se calculează cu relația:

$$N_{linii} \geq \log_2 N_{ZONE} = \log_2 4 \Rightarrow 2 \text{ linii}$$

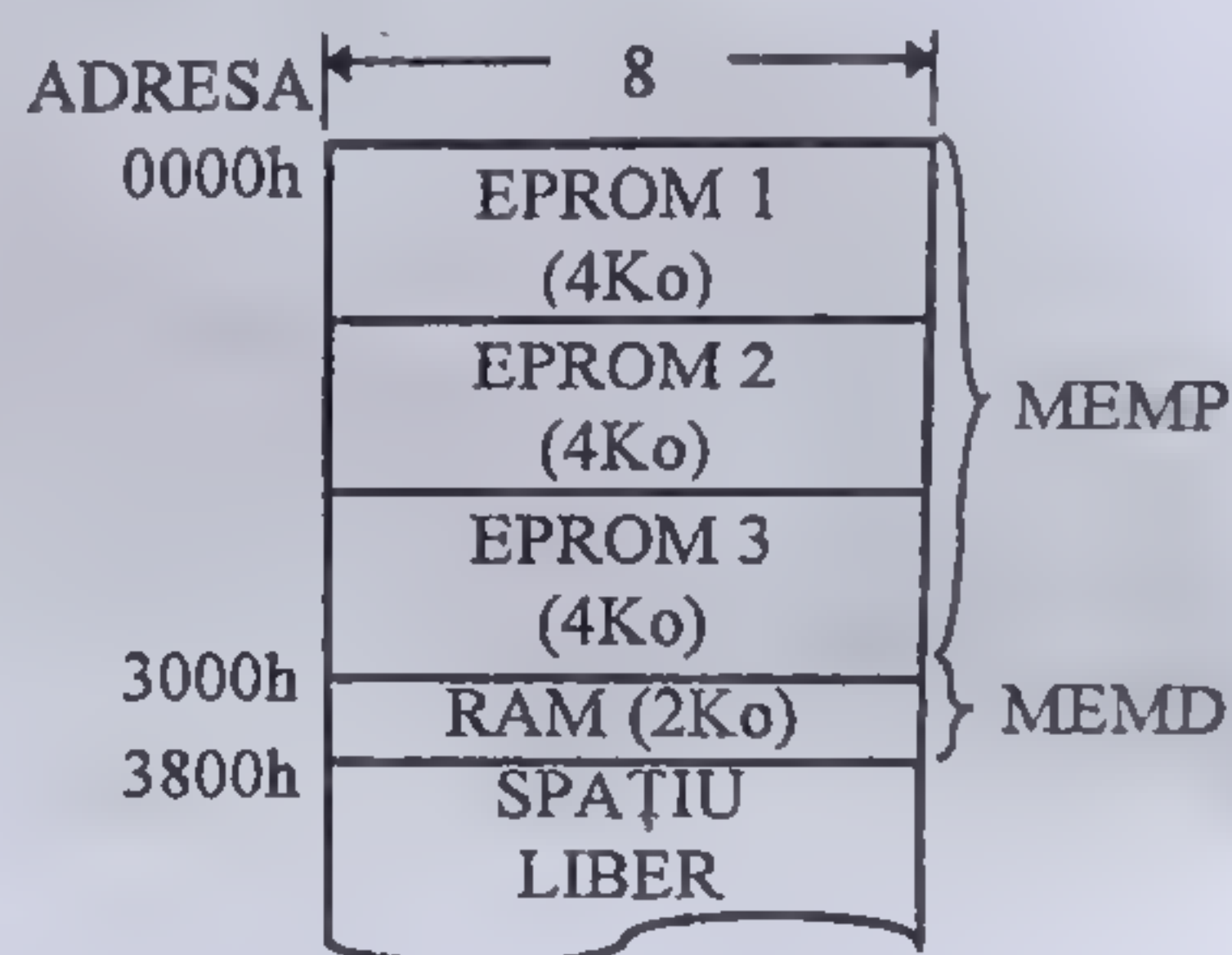


Fig.1.14. Harta memoriei la exemplul 1.2

Tabelul 1.1.

Tipul memoriei	Zona de memorie	Linii folosite pentru adresarea internă	Linii pentru selecție							
			liniară				cu decodificare			
		$A_0 \dots A_{10} A_{11}$	A_{12}	A_{13}	A_{14}	A_{15}	A_{12}	A_{13}	A_{14}	A_{15}
EPROM 1	0000 ÷ 0FFFh	x x x	1	0	0	0	0	0	0	0
EPROM 2	1000 ÷ 1FFFh	x x x	0	1	0	0	1	0	0	0
EPROM 3	2000 ÷ 2FFFh	x x x	0	0	1	0	0	1	0	0
RAM	3000 ÷ 37FFh	x x 0	0	0	0	1	1	1	0	0

5) Se vor considera ambele metode de selecție:

a) Conectarea la magistrale a circuitelor de memorie în cazul selecției liniare este dată în schema din fig.1.15. Se observă că, în corespondență cu tab.1.1, este necesar un nivel de porți inversoare pentru selecție, întrucât circuitele de memorie au \overline{CS} activ pe nivel coborât.

De asemenea, unele dispozitive de memorie EPROM dispun de o linie de comandă a ieșirilor - \overline{OE} (Output Enable), care poate fi comandată de linia \overline{MEMR} a microprocesorului. În cazul în care această linie lipsește, este suficientă selecția prin liniile \overline{CS} .

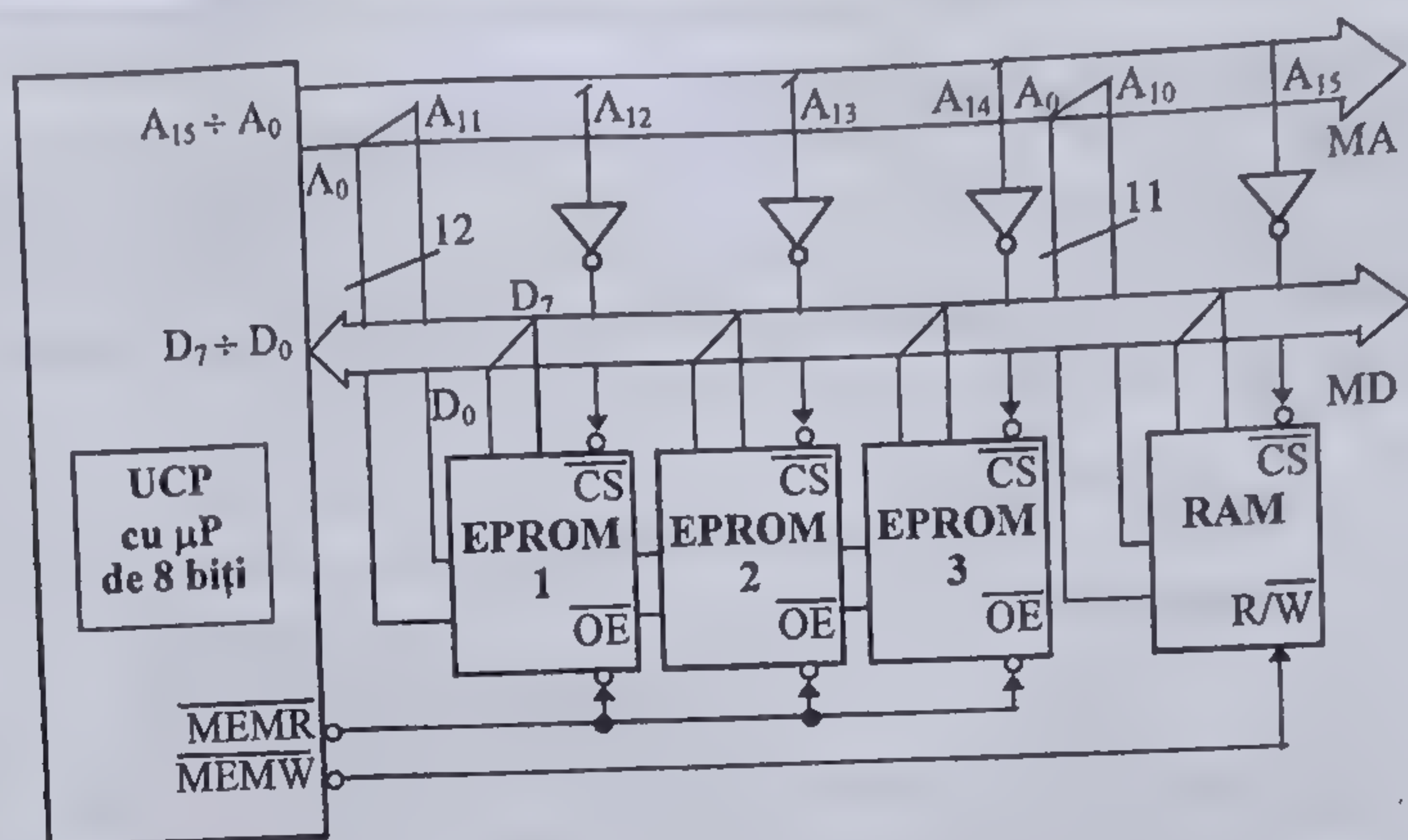


Fig.1.15. Selecția liniară a circuitelor de memorie

b) În cazul selecției cu decodificare este necesar un DCD/DMUX 2/4. În fig.1.16a este prezentată schema de conectare, în care se utilizează un decodificator cu ieșiri active pe nivel coborât (de ex. SN74LS42). Aici trebuie menționat faptul că se realizează o selecție ciclică atunci când se activează prin program liniile A_{14} sau A_{15} , ceea ce poate constitui un dezavantaj.

Folosirea unui demultiplexor prezintă avantajul că se poate separa spațiul de memorie de alte spații, cum ar fi spațiul I/E. Dacă una din liniile de validare ale DMUX, spre exemplu \overline{E}_1

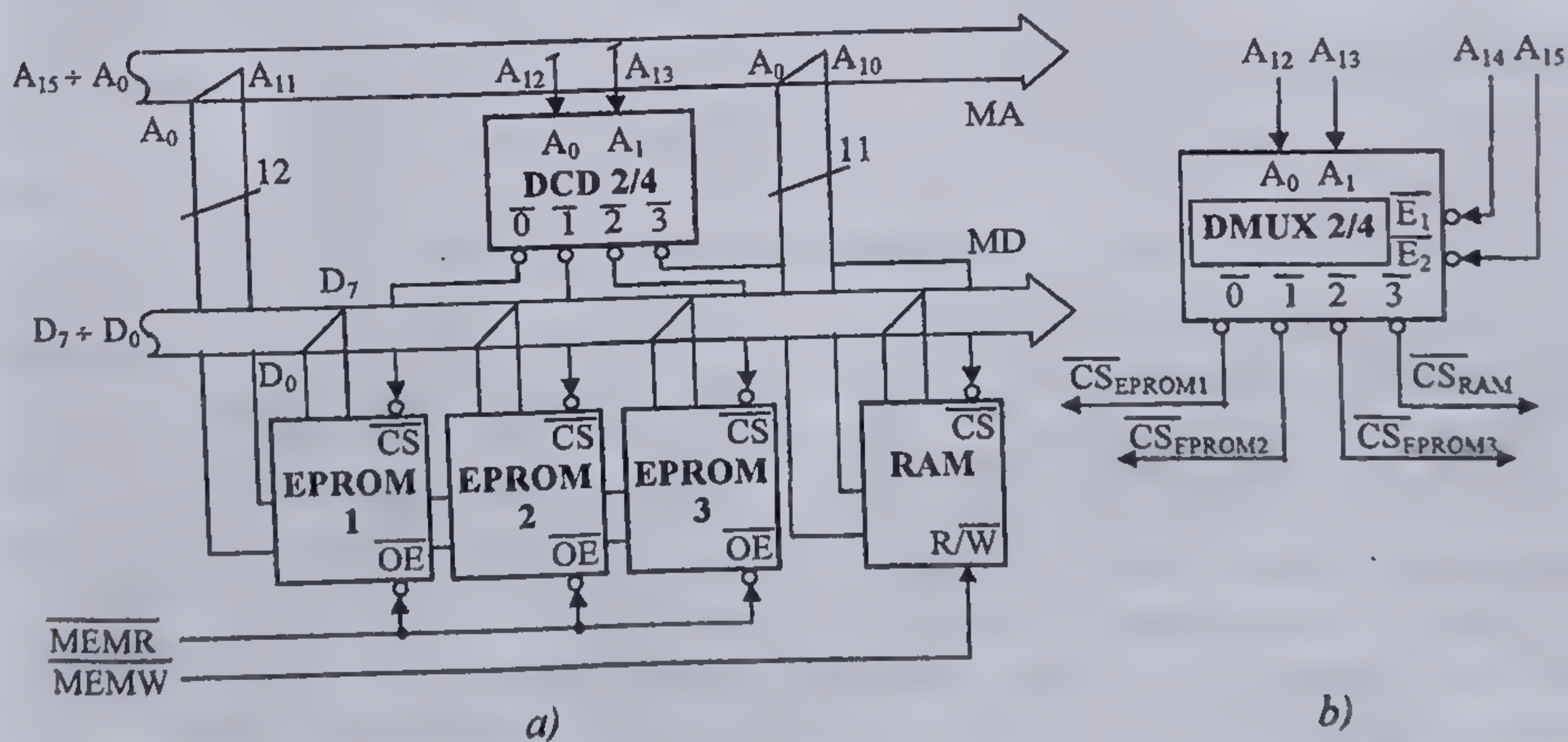


Fig.1.16. Selecția cu decodificare (a) și cu demultiplexare (b) a circuitelor de memorie

(Enable), este conectată la următoarea linie de adresă neutilizată, A_{14} , iar linia \overline{E}_2 la A_{15} , se asigură o protecție a primei zone de 16Ko de memorie (fig.1.16b). Unele microprocesoare posedă linii distincte pentru controlul spațiilor de memorie și de I/E, care pot fi folosite la comanda demultiplexoarelor de selecție.

1.1.2.3. Conectarea memoriilor RAM dinamice (DRAM)

Citirea și înscrierea memoriilor DRAM

Spre deosebire de memoriile RAM statice, circuitele de memorie RAM dinamică sunt prevăzute cu semnale suplimentare de comandă, care derivă din modul de organizare internă și din tehnologia de fabricație folosită. În fig.1.17 se prezintă structura generală a logicii de acces la memoria DRAM. În continuare se va analiza rolul circuitelor și a semnalelor de comandă specifice pentru realizarea accesului la blocul de memorie DRAM.

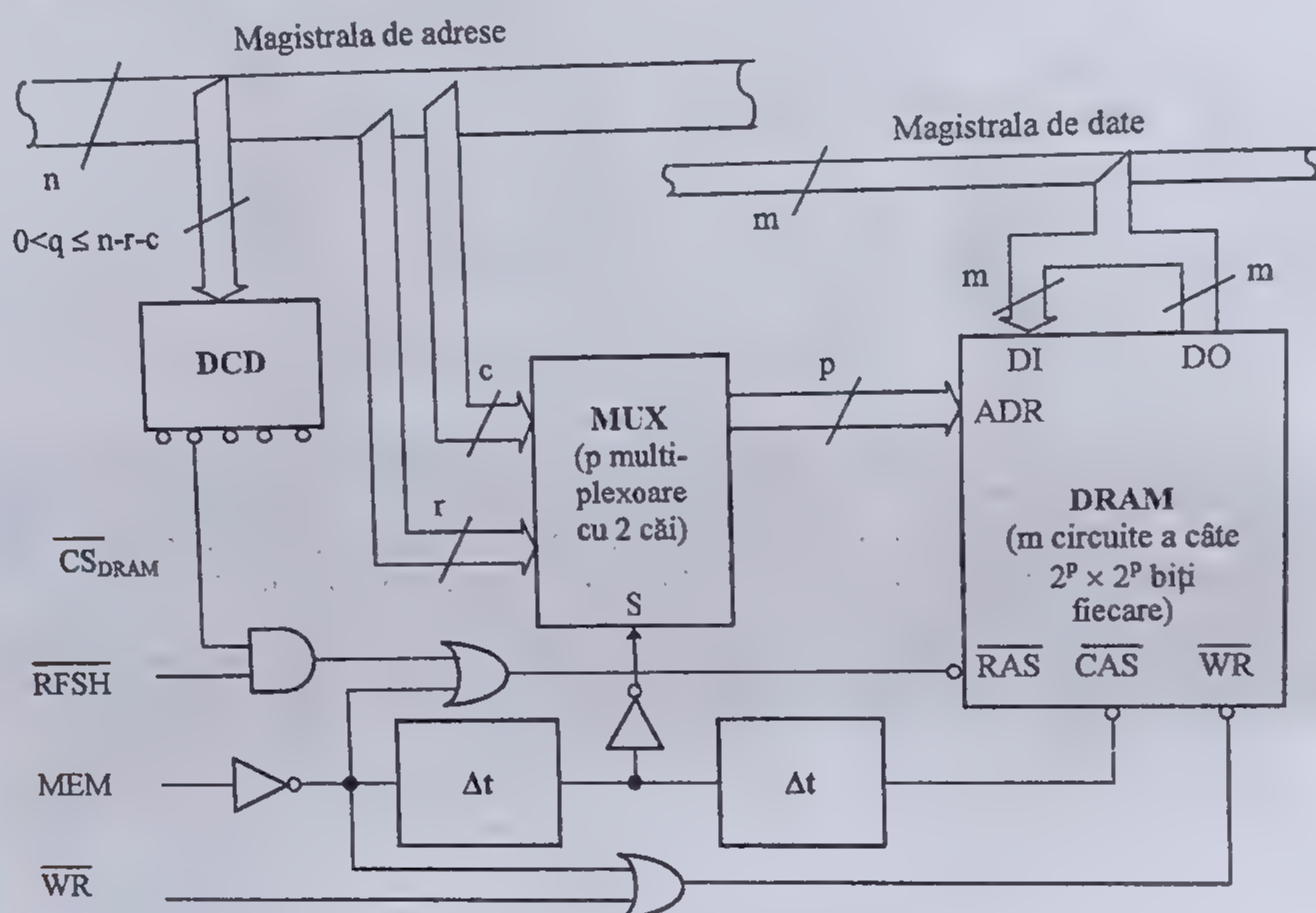


Fig.1.17. Conectarea memoriilor RAM dinamice

Uzual, un circuit DRAM are lungimea cuvântului de 1 bit. El conține unul sau mai multe bancuri de memorie, fiecare banc fiind realizat sub forma unei matrici cu 2^r rânduri și 2^c coloane. La intersecția unui rând cu o coloană se află o celulă de memorie, realizată fizic sub forma unui microcondensator. Prin cele două stări electrice - încărcat și descărcat - acesta codifică stările logice "1" și respectiv "0". Accesul la o astfel de celulă se realizează prin selecția liniei și coloanei corespunzătoare, folosind o adresă de rând (r linii) și o adresă de coloană (c linii), deci în total $r+c$ linii de adresă. Aceste adrese sunt decodificate separat, iar cele două semnale de selecție care rezultă trebuie aplicate succesiv, mai întâi pentru activarea rândului și apoi, după un interval de timp de cca. 40 ns, pentru activarea coloanei, la intersecția cărora se află celula accesată.

Decodificatoarele de linie și de coloană sunt integrate pe cipul de memorie, dar adresa de linie și cea de coloană trebuie aplicate succesiv, pe cele p linii de adresă ale circuitului DRAM (de obicei $r=c=p$), respectându-se întârzierea menționată anterior. Aceasta implică multiplexarea în exterior cu ajutorul unor circuite de tip multiplexor cu 2 căi, controlate pe intrarea S.

În schema din fig.1.17 apar semnale de comandă specifice, neîntâlnite la schemele cu memorii RAM statice (SRAM), și anume:

- \overline{RFSH} (ReFreSH) - pentru comanda reîmprospătării informației memorate;

Capitolul 1 - Structura sistemelor cu microprocesoare

- MEM - pentru activarea dialogului cu memoria, atât la citire, cât și la înscriere, deci de tip MEMR+MEMW (v.fig.1.9).

În momentul depunerii adresei de memorie pe magistrala de adrese, $S=0$ și pe intrările de adresă (ADR) ale memoriei DRAM ajung cele mai puțin semnificative r linii ale acesteia (adresa de rând). La activarea semnalului de acces la memorie ($MEM=1$), dacă este selectată memoria DRAM ($\overline{CS}_{DRAM}=0$, provenit din decodificarea liniilor superioare ale magistralei de adrese), se activează semnalul \overline{RAS} (Row Address Strobe). Pe frontul descrescător, acest semnal determină memorarea adresei de rând în DRAM, după care validează decodificatoarele de linie.

După un interval de timp Δt , intrarea de selecție a multiplexorului se schimbă ($S=1$) și pe liniile ADR ale memoriei DRAM apare adresa de coloană, formată din următoarele c linii de adresă mai semnificative. După un alt interval de timp Δt , necesar pentru comutarea liniilor de ieșire ale multiplexorului și pentru stabilizarea adresei de coloană, este activat și semnalul \overline{CAS} (Column Address Strobe). Pe frontul descrescător, acest semnal de comandă determină memorarea adresei de coloană în DRAM, după care validează decodificatoarele de coloană și tamponarele de intrare/ieșire date, de pe liniile DI/DO.

Într-un ciclu de citire (fig.1.18), linia de intrare \overline{WR} a memoriei DRAM rămâne dezactivată. La un interval de timp $t_{CAccess}$ de la activarea liniei \overline{CAS} , datele apar pe liniile de ieșire DO ale memoriei DRAM și pot fi citite de unitatea centrală.

Într-un ciclu de scriere sunt posibile următoarele două situații:

- dacă linia \overline{WR} este activată înaintea lui \overline{CAS} (ca în fig.1.19), se obține un *ciclu de scriere în avans* (early write cycle). În acest caz, liniile de ieșire date ale memoriei DRAM, DO, rămân în starea de înaltă impedanță (HZ) pe durata întregului ciclu de scriere. Înscrierea datei se realizează pe frontul descrescător al semnalului \overline{CAS} , cu timpi de prestabilire și de menținere raportați la acest semnal. Acest tip de scriere permite conectarea directă a liniilor DI și DO la magistrala de date, fără circuite suplimentare, de separare a sensului de transfer.

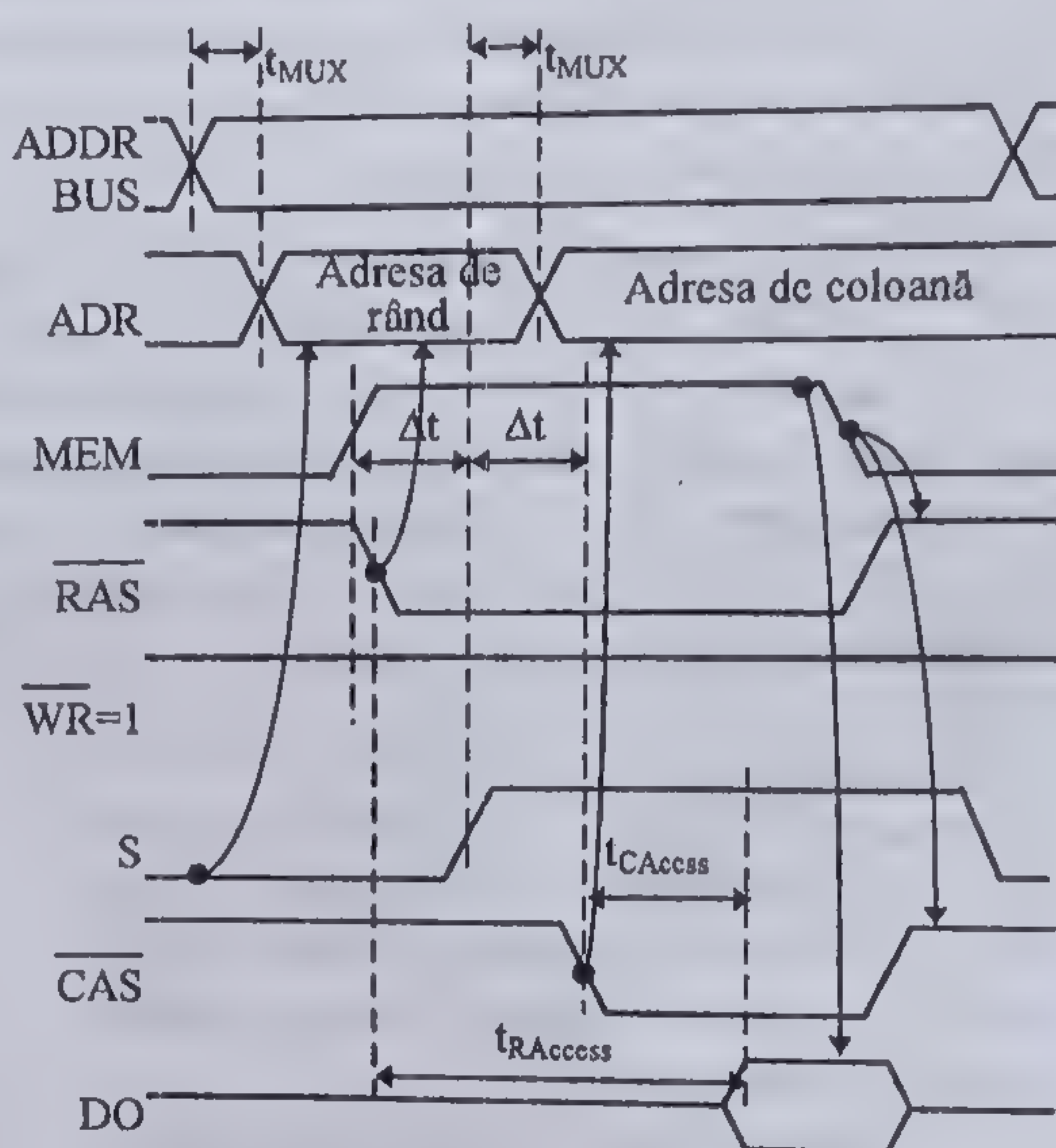


Fig.1.18. Ciclul de citire a memoriei DRAM

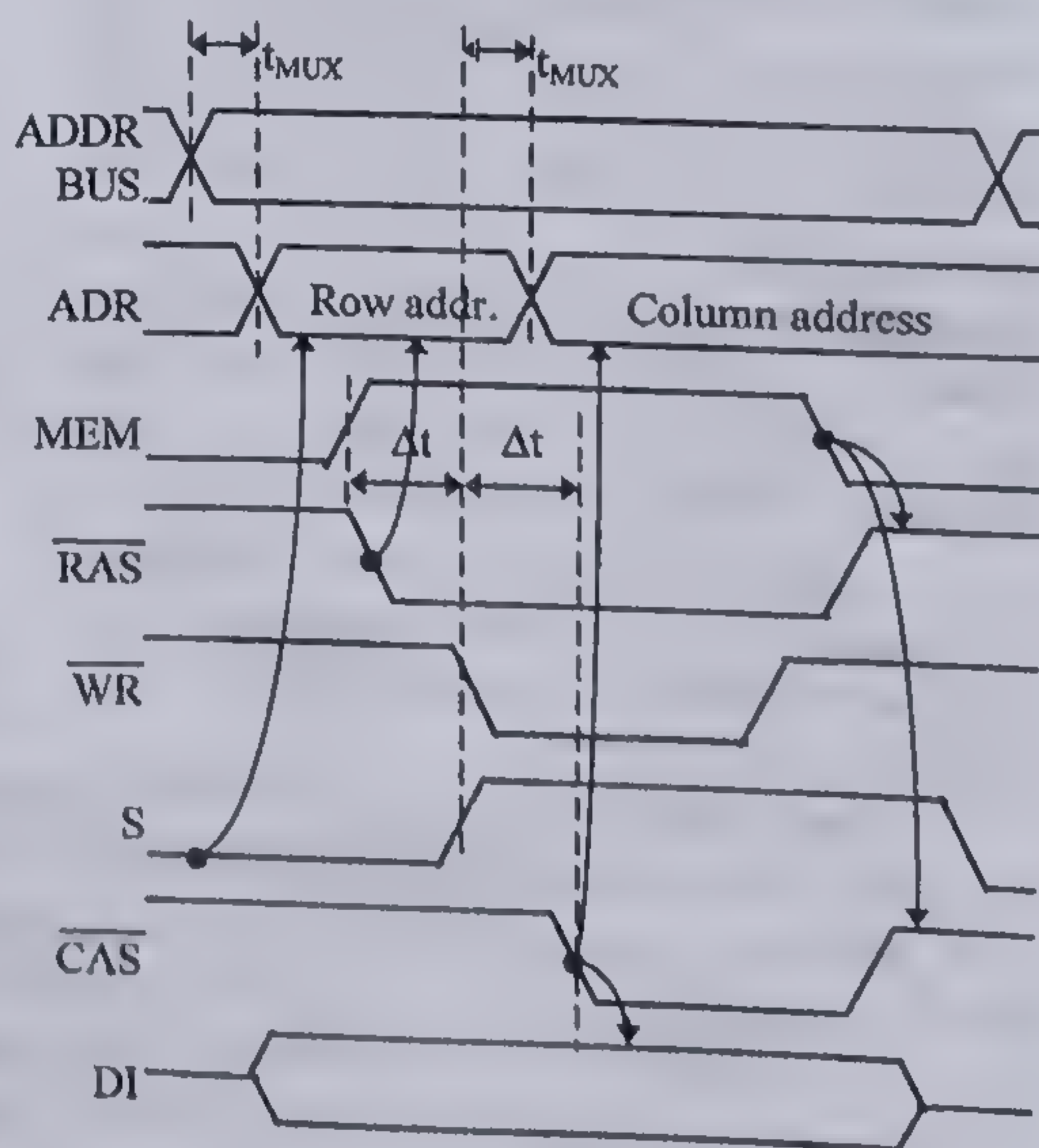


Fig.1.19. Ciclul de scriere în avans a memoriei DRAM

- dacă linia \overline{WR} este activată după activarea liniei \overline{CAS} , se obține un *ciclu de scriere întârziat* (delayed write cycle). În acest caz, în intervalul de timp de la activarea liniei \overline{CAS} și până la activarea liniei \overline{WR} , ieșirile vor furniza data ca și într-un ciclu de citire obișnuit. Pentru ca data de pe liniile DO să nu ajungă pe magistrala de date, unde este prezentă deja data ce trebuie înscrisă în DRAM, sunt necesare măsuri suplimentare de protecție, prin utilizarea unor circuite amplificatoare separatoare de sens (de exemplu 8216). Data furnizată de unitatea centrală va fi înscrisă în memoria DRAM pe frontul descrescător al semnalului \overline{WR} , cu timpi de prestabilire și de menținere raportați la acest semnal. Acest tip de ciclu se mai numește ciclu de citire-modificare-scriere (read-modify-write cycle).

Reîmprospătarea memoriilor RAM dinamice

În afară de ciclurile de citire și scriere, prezentate mai sus, memoriile RAM dinamice necesită un acces periodic pentru prevenirea pierderii informației; aceasta datorită descărcării în timp a microcondensatoarelor care formează celulele de memorie. Astfel, fiecare celulă trebuie să fie activată periodic, de regulă la intervale de cel puțin 2 ms, ceea ce permite refacerea potențialelor armăturilor microcondensatoarelor, operație care poartă numele de *reîmprospătare* sau *refreșare* (refresh). Circuitele de memorie RAM dinamică sunt astfel realizate încât nu este necesară activarea individuală a celulelor de memorie, fiind suficientă selectarea unui rând pentru ca toate celulele de pe rândul respectiv să-și reîmprospăteze informația. Drept consecință, adresa de reîmprospătare va avea numai r biți, iar timpul necesar acestei operații se reduce drastic.

În fig.1.20 se prezintă diagrama de timp a unei operații de reîmprospătare. Adresa de reîmprospătare se depune pe cele mai puțin semnificative r linii ale magistralei de adrese, ajungând la intrările de adresă ADR ale DRAM, după care se activează semnalul \overline{RAS} . El determină reîmprospătarea informației de pe rândul selectat de adresa de reîmprospătare, adresă care este apoi incrementată și este folosită în mod similar la refreșarea următorului rând ș.a.m.d. Prin menținerea liniei \overline{CAS} în "1" se evită orice ieșire de date, obținându-se astfel și o importantă reducere a puterii consumate (v. *Exemplul 1.3* și fig.1.21).

Reîmprospătarea memoriei DRAM trebuie să se realizeze în mod continuu, sistematic și în paralel cu accesul aleator pentru citire sau scriere. Deoarece în ambele cazuri se utilizează atât liniile de adresă ADR cât și semnalul \overline{RAS} , pentru evitarea conflictelor se folosește o logică de refreșare special proiectată, cu arbitrarea accesului concurențial la memoria DRAM. Ea este cea care, în perioadele în care magistrala de adrese este liberă, va depune pe cele mai puțin semnificative r linii adresa de refreșare și va activa semnalul \overline{RFSH} (fig.1.17). În unele cazuri reîmprospătarea este asigurată chiar de către microprocesor. Este și cazul microprocesorului Z80, care înglobează pe cip un automat de refreșare. Aceasta simplifică extrem de mult conectarea

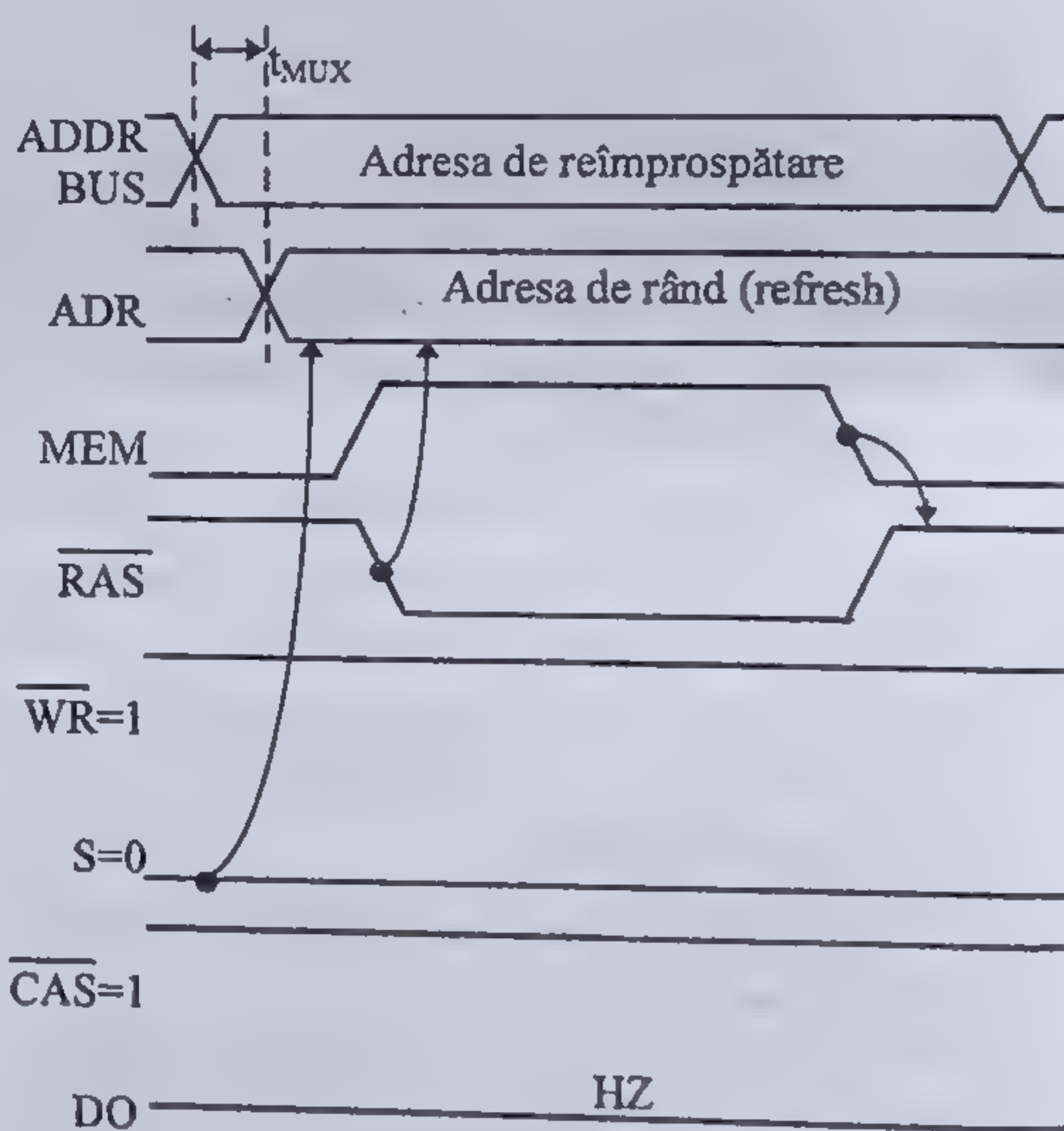


Fig.1.20. Ciclul de reîmprospătare a memoriei DRAM

memoriilor RAM dinamice în sistemele cu microprocesoare Z80, așa cum se va exemplifica în §3.4.2.

Proiectarea blocului de memorie de date cu dispozitive DRAM

Dacă se păstrează caracterul aleator al accesului la memoria DRAM, apar întârzieri datorate selecției, atât a rândului, cât și a coloanei, după cum s-a văzut anterior. Pentru a crește viteza de acces se poate utiliza modul de lucru pe pagină. Spre exemplu, se poate menține aceeași adresă de rând și se modifică numai adresa de coloană, eliminându-se astfel timpul necesar pentru prestabilirea și memorarea adreselor de linie.

Întârzierile Δt care apar în fig.1.17 pot fi obținute prin sincronizarea cu semnalul de tact al unității centrale și prin utilizarea unor porți logice, după cum se va vedea în continuare. Dacă sistemul conține mai multe blocuri de memorie RAM dinamică se poate utiliza schema cu semnal $\overline{\text{CAS}}$ comun din fig.1.17 sau o schemă cu semnal $\overline{\text{RAS}}$ comun, ca în exemplul următor.

Exemplul 1.3. Se consideră cazul unui sistem cu microprocesor de 8 biți, care necesită o memorie de date (MEMD) de 32 Kocteți. Se va realiza această MEMD din două blocuri de memorie DRAM de 16 Kocteți, realizate cu circuite de tip 4116, cu capacitatea de 16 Kbiți fiecare (fig.1.21). Harta memoriei, prezentată în tab.1.2, pune în evidență spațiul de adrese ocupat de

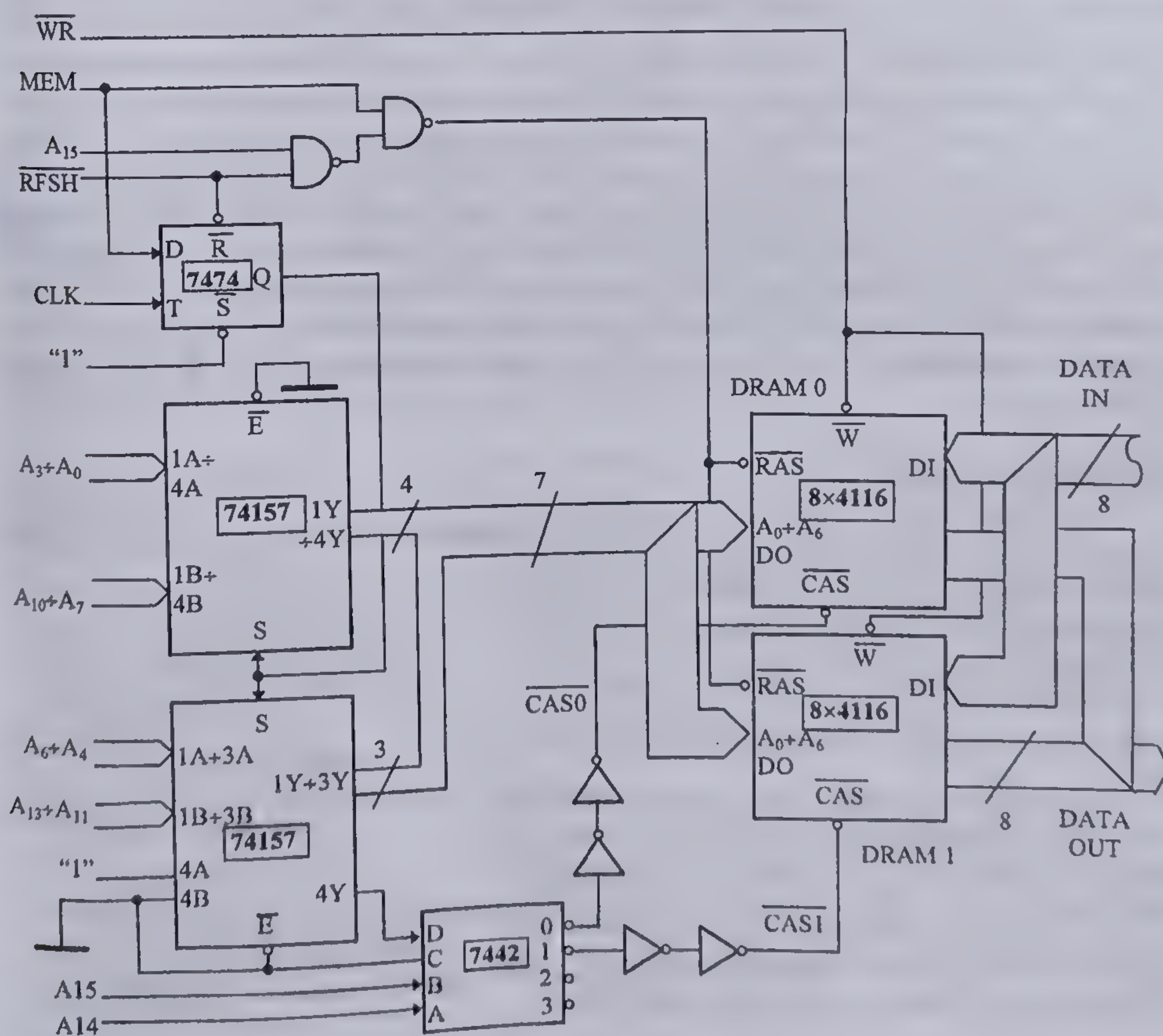


Fig.1.21. Conectarea a două blocuri de 16 Ko DRAM realizate cu circuite 4116

fiecare din cele două blocuri de memorie, DRAM0 și DRAM1 și permite sinteza logicii de decodificare și de generare a semnalelor de selecție, $\overline{\text{CAS0}}$ și respectiv $\overline{\text{CAS1}}$.

Tabelul 1.2.

Bloc DRAM	A15	A14	A13 ÷ A0	$\overline{\text{CAS0}}$	$\overline{\text{CAS1}}$	Spațiu de adrese
DRAM0	0	0	x	0	1	0000h ÷ 3FFFh
DRAM1	0	1	x	1	0	4000h ÷ 7FFFh
-	1	x	x	1	1	8000h ÷ FFFFh (neutilizat)

Sunt necesare câte 8 circuite 4116 pentru fiecare bloc, care vor fi citite și înscrise simultan pentru a se obține un cuvânt de date cu lungimea de 8 biți. Pentru multiplexarea liniilor de adresă se folosesc 2 multiplexoare de tip SN74157, care conțin fiecare câte 4 multiplexoare cu două căi. Șapte din cele opt multiplexoare furnizează memoriei DRAM adresa de rând ($A_6 \div A_0$), când linia de selecție comună $S=0$ și, ulterior, adresa de coloană ($A_{13} \div A_7$), când S devine "1". Cel de-al 8-lea multiplexor are rolul de a valida generarea semnalului $\overline{\text{CAS}}$, corespunzător blocului de memorie selectat. Selecția se realizează cu ajutorul liniilor superioare de adresă A_{15} și A_{14} , decodificate cu un circuit SN7442.

Semnalul $\overline{\text{RAS}}$ este comun pentru ambele blocuri de memorie și este generat doar în timpul unui ciclu de citire/scriere din/în memoria DRAM (când $\text{MEM}=1$ și $A_{15}=0$) sau în timpul unui ciclu de reîmprospătare a memoriei DRAM (când $\text{MEM}=1$ și $\overline{\text{RFSH}}=0$). Semnalul MEM este furnizat de unitatea centrală, iar semnalul $\overline{\text{RFSH}}$ este generat de logica de reîmprospătare. Multiplexarea adresei DRAM este sincronizată cu frontul crescător al semnalului de tact al unității centrale, CLK, cu ajutorul unui bistabil de tip D (SN7474), realizându-se astfel întârzierea dintre activarea semnalelor MEM și S (primul interval Δt din fig.1.18 și 1.19). Cel de-al doilea interval Δt este obținut prin însumarea timpului de multiplexare a adresei cu cel de comutare al decodicatorului și cu timpii de propagare a semnalelor $\overline{\text{CAS0}}$, respectiv $\overline{\text{CAS1}}$, prin cele două inversoare, înainte de a ajunge la blocurile de memorie corespunzătoare. În timpul unui ciclu de refreșare este inhibată activarea semnalelor $\overline{\text{CAS}}$, prin resetarea bistabilului de către semnalul $\overline{\text{RFSH}}$.

1.1.2.4. Memoria stivă

Datorită numărului redus de registre interne ale unui microprocesor, în multe situații este necesară salvarea temporară a conținutului acestora. Astfel de situații sunt specifice lucrului cu subrutine, atât la realizarea unui program normal, cât și în regimul de întreruperi. Unele microprocesoare (de exemplu 8008) au o mică memorie RAM internă pentru astfel de salvări, dar în general se folosește memoria externă de date.

Pentru ca aceste informații să nu se suprapună peste datele programului, pentru salvări se folosește de regulă zona situată la capătul superior al memoriei de date (fig.1.22). Această zonă poartă denumirea de *memorie stivă* (stack memory) sau pe scurt *stivă*, deoarece aici se depozitează conținutul registrelor, unul peste altul. Spre deosebire de memoria de date, care se

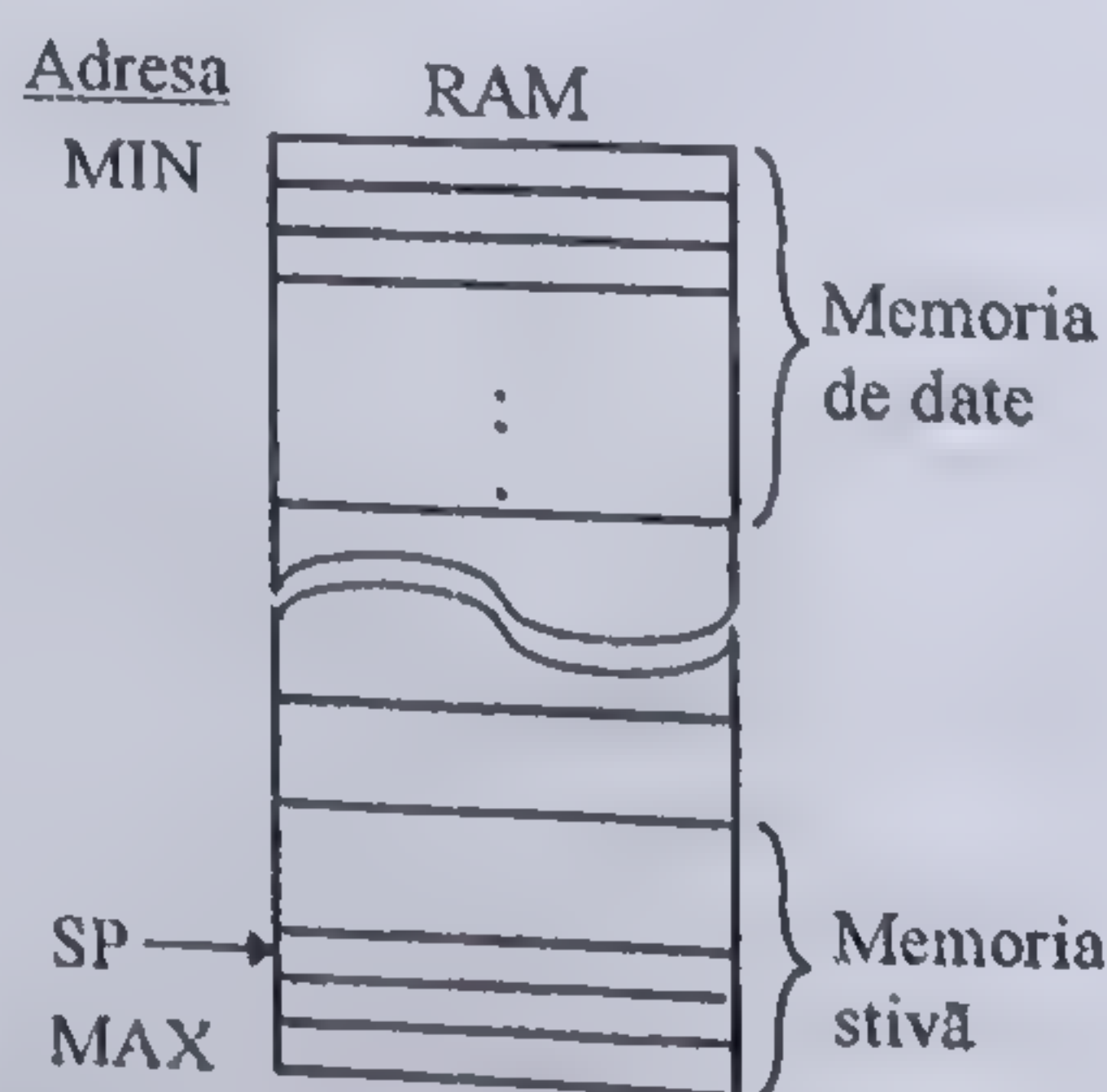


Fig.1.22. Localizarea memoriei stivă

organizează în sensul crescător al adreselor, stiva se organizează de la adrese mari spre adrese mici; de aceea se spune că stiva "crește în jos".

Pentru indicarea primei adrese libere în stivă, microprocesoarele posedă un registru special, denumit *indicator al vârfului stivei* sau SP (Stack Pointer). Acest "pointer" are dimensiunea identică cu a magistralei de adrese, pentru a putea specifica adresa completă. Datorită importanței pe care o prezintă la structurarea programelor, microprocesoarele posedă instrucțiuni speciale, de inițializare a SP și de lucru cu stiva.

1.1.3. Porturi de intrare-ieșire

Porturile I/E sunt circuite prin intermediul cărora se realizează un transfer de informație între microprocesor și exterior, altceva decât memoria. Sunt realizate astfel încât să poată fi conectate direct la magistrala sistemului și să poată dialoga nemijlocit cu microprocesorul unității centrale. Aceste dispozitive au rolul de interfațare a sistemelor cu microprocesoare cu periferice informatice standard, cu interfețe de proces sau cu orice alt dispozitiv care poate transmite sau poate primi informații de la UCP (mai puțin memoria), generic denumite *dispozitive I/E*. Porturile de interfață I/E au rolul de adaptare și sincronizare a transferului de informație dintre UCP și periferice.

1.1.3.1. Transferul de informație dintre UCP și porturile I/E

În acest caz, transferul de informație se realizează prin execuția ciclurilor de scriere într-un port de ieșire, respectiv de citire dintr-un port de intrare.

Înscrierea datelor într-un port de ieșire

Pentru început, se poate considera că datele provin de la o sursă generică de informație, conectată la cele 3 magistrale: de adrese, de date și de comandă. Orice port de ieșire are asociată o adresă, similară cu cea a unei locații de memorie. Într-o formă simplificată, diagrama temporală de scriere într-un port de ieșire, PE, este prezentată în fig.1.23.

Important este ca adresa și datele să fie stabile înainte ca pe linia de comandă să fie activat impulsul de scriere $\overline{I/O\overline{W}}$, cu durata corespunzătoare (T_{WP}). Se observă distincția care se face între memorie și un port de ieșire, prin semnale de comandă specifice: \overline{MEMW} pentru memorie și $\overline{I/O\overline{W}}$ pentru portul de ieșire. Microprocesoarele, după cum s-a văzut, au linii de comandă distincte pentru spațiul I/E, dar există și cazuri (de exemplu M6800) când porturile I/E sunt tratate la fel ca și locațiile de memorie, cu aceleași linii de comandă.

Pentru înscrierea unei date din registrul R al unui microprocesor într-un port de ieșire, PE, de adresă β , rezultă următoarea secvență de operații:

a) pregătirea transferului

$$MA \leftarrow \beta$$

$$MD \leftarrow (R)$$

$$MC \leftarrow \overline{I/O\overline{W}}$$

} informații transmise succesiv de microprocesor

b) scrierea în portul de ieșire

$$PE(\beta) \leftarrow (MD) \quad \text{conținutul MD se transferă în PE de adresă } \beta$$

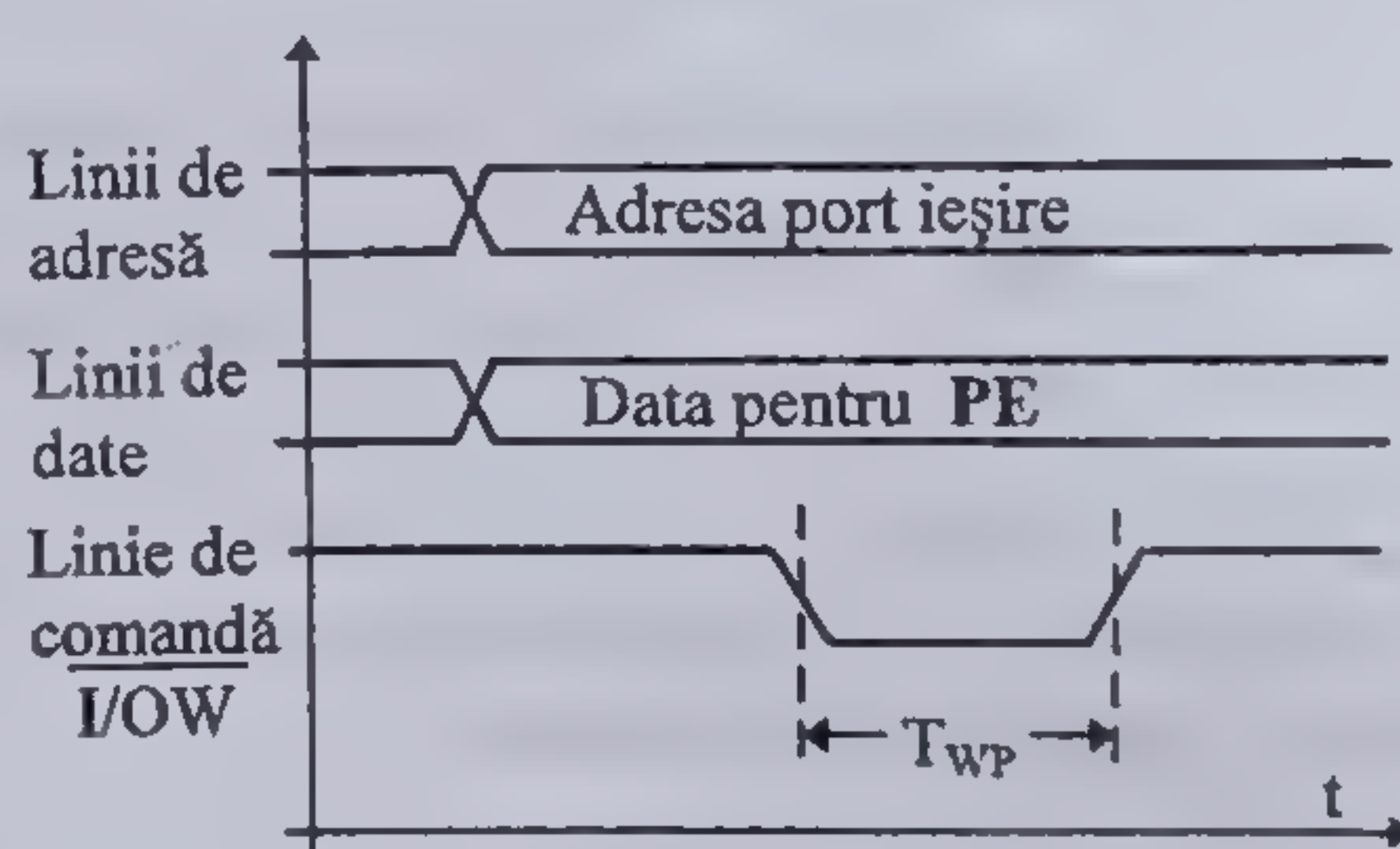


Fig.1.23. Diagrama simplificată la scrierea într-un port de ieșire

Se observă similitudinea cu scrierea datelor într-o locație de memorie, descrisă în §1.1.2.1.

Citirea datelor de la un port de intrare

Un dispozitiv de intrare poate fi privit ca o sursă de informație, altă decât memoria, capabilă să transmită date spre microprocesor prin intermediul unui port de intrare, PI. Data disponibilă în portul de intrare poate fi preluată de către microprocesor într-un ciclu de citire a PI. O diagramă temporală simplificată, care arată relațiile dintre semnalele de pe magistrala sistemului în acest caz, este prezentată în fig.1.24.

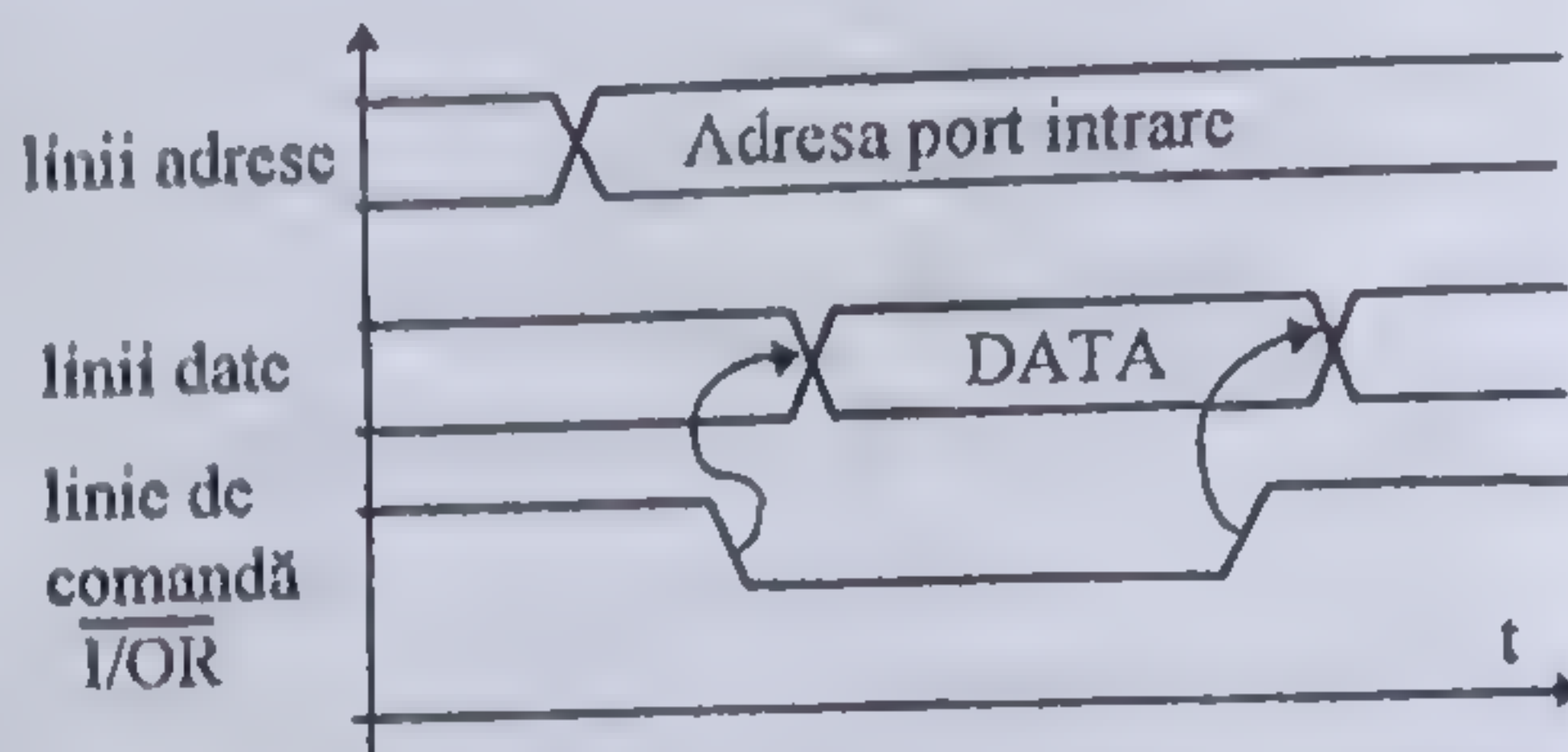


Fig.1.24. Diagrama simplificată la citirea unui port de intrare

De notat că adresa portului de intrare trebuie să existe pe MA înainte de activarea semnalului de citire, $\overline{I/OR}$. Pe durata cât acest semnal este activ, datele sunt depuse de PI pe magistrala de date. În acest timp, microprocesorul citește informația de pe magistrala de date. Dacă portul de intrare PI are adresa γ , atunci transferul se realizează prin următoarea secvență de operații:

a) pregătirea portului de intrare

$$\left. \begin{array}{l} MA \leftarrow \gamma \\ MC \leftarrow \overline{I/OR} \end{array} \right\} \text{informații transmise succesiv, de către microprocesor}$$

b) transferul informației în registrul R al microprocesorului

$$\left. \begin{array}{l} MD \leftarrow PI(\gamma) \\ R \leftarrow (MD) \end{array} \right\} \text{citirea de către microprocesor a informației de pe MD}$$

Și în acest caz este evidentă similitudinea derulării procedurilor de citire dintr-un port și din memorie, ceea ce justifică posibilitatea de tratare identică în cazul porturilor I/E mapate în spațiul de adresare a memoriei.

1.1.3.2. Conectarea porturilor I/E la magistrala sistemului

Deoarece dialogul microprocesorului cu perifericele se efectuează prin intermediul porturilor I/E, realizarea corespunzătoare a transferului de informație depinde de corecta conectare a acestora la magistrale. Din cele prezentate în paragraful precedent rezultă că porturile I/E, care intermediază acest transfer, trebuie să posede următoarele caracteristici:

- să poată fi selectate/deselectate individual de către microprocesor;
- să asigure o dimensiune corespunzătoare a cuvintelor transferate de la/spre μP ;
- să permită memorarea temporară a informației de la/spre UCP, în vederea sincronizării dintre microprocesor și dispozitivele I/E;
- să fie compatibile din punct de vedere al încărcării (fan in, fan out) cu caracteristicile electrice ale magistralelor.

În prezent, producătorii de microprocesoare realizează atât dispozitive I/E simple, sub forma unor *porturi I/E de uz general*, cât și structuri complexe, programabile, specializate pentru anumite aplicații (controlere DMA, de întreruperi, de comunicație, timere etc.). Indiferent de complexitatea lor, microprocesorul tratează aceste dispozitive în mod unitar, prin porturile sau registrele înglobate în structura lor, având posibilitatea de a selecta/deselecta individual fiecare port și de a efectua cu el aceleași operații ca și cu un port I/E de uz general. În plus, la dispozitivele

Se observă similitudinea cu scrierea datelor într-o locație de memorie, descrisă în §1.1.2.1.

Citirea datelor de la un port de intrare

Un dispozitiv de intrare poate fi privit ca o sursă de informație, altă decât memoria, capabilă să transmită date spre microprocesor prin intermediul unui port de intrare, PI. Data disponibilă în portul de intrare poate fi preluată de către microprocesor într-un ciclu de citire a PI. O diagramă temporală simplificată, care arată relațiile dintre semnalele de pe magistrala sistemului în acest caz, este prezentată în fig.1.24.

De notat că adresa portului de intrare trebuie să existe pe MA înainte de activarea semnalului de citire, $\overline{I/OR}$. Pe durata cât acest semnal este activ, datele sunt depuse de PI pe magistrala de date. În acest timp, microprocesorul citește informația de pe magistrala de date. Dacă portul de intrare PI are adresa γ , atunci transferul se realizează prin următoarea secvență de operații:

a) pregătirea portului de intrare

$$\left. \begin{array}{l} MA \leftarrow \gamma \\ MC \leftarrow \overline{I/OR} \end{array} \right\} \text{informații transmise succesiv, de către microprocesor}$$

b) transferul informației în registrul R al microprocesorului

$$\left. \begin{array}{l} MD \leftarrow PI(\gamma) \\ R \leftarrow (MD) \end{array} \right\} \text{citirea de către microprocesor a informației de pe MD}$$

Și în acest caz este evidentă similitudinea derulării procedurilor de citire dintr-un port și din memorie, ceea ce justifică posibilitatea de tratare identică în cazul porturilor I/E mapate în spațiul de adresare a memoriei.

1.1.3.2. Conectarea porturilor I/E la magistrala sistemului

Deoarece dialogul microprocesorului cu perifericele se efectuează prin intermediul porturilor I/E, realizarea corespunzătoare a transferului de informație depinde de corecta conectare a acestora la magistrale. Din cele prezentate în paragraful precedent rezultă că porturile I/E, care intermediază acest transfer, trebuie să posede următoarele caracteristici:

- să poată fi selectate/deselectate individual de către microprocesor;
- să asigure o dimensiune corespunzătoare a cuvintelor transferate de la/spre μP ;
- să permită memorarea temporară a informației de la/spre UCP, în vederea sincronizării dintre microprocesor și dispozitivele I/E;
- să fie compatibile din punct de vedere al încărcării (fan in, fan out) cu caracteristicile electrice ale magistralelor.

În prezent, producătorii de microprocesoare realizează atât dispozitive I/E simple, sub forma unor *porturi I/E de uz general*, cât și structuri complexe, programabile, specializate pentru anumite aplicații (controlere DMA, de întreruperi, de comunicație, timere etc.). Indiferent de complexitatea lor, microprocesorul tratează aceste dispozitive în mod unitar, prin porturile sau registrele înglobate în structura lor, având posibilitatea de a selecta/deselecta individual fiecare port și de a efectua cu el aceleași operații ca și cu un port I/E de uz general. În plus, la dispozitivele

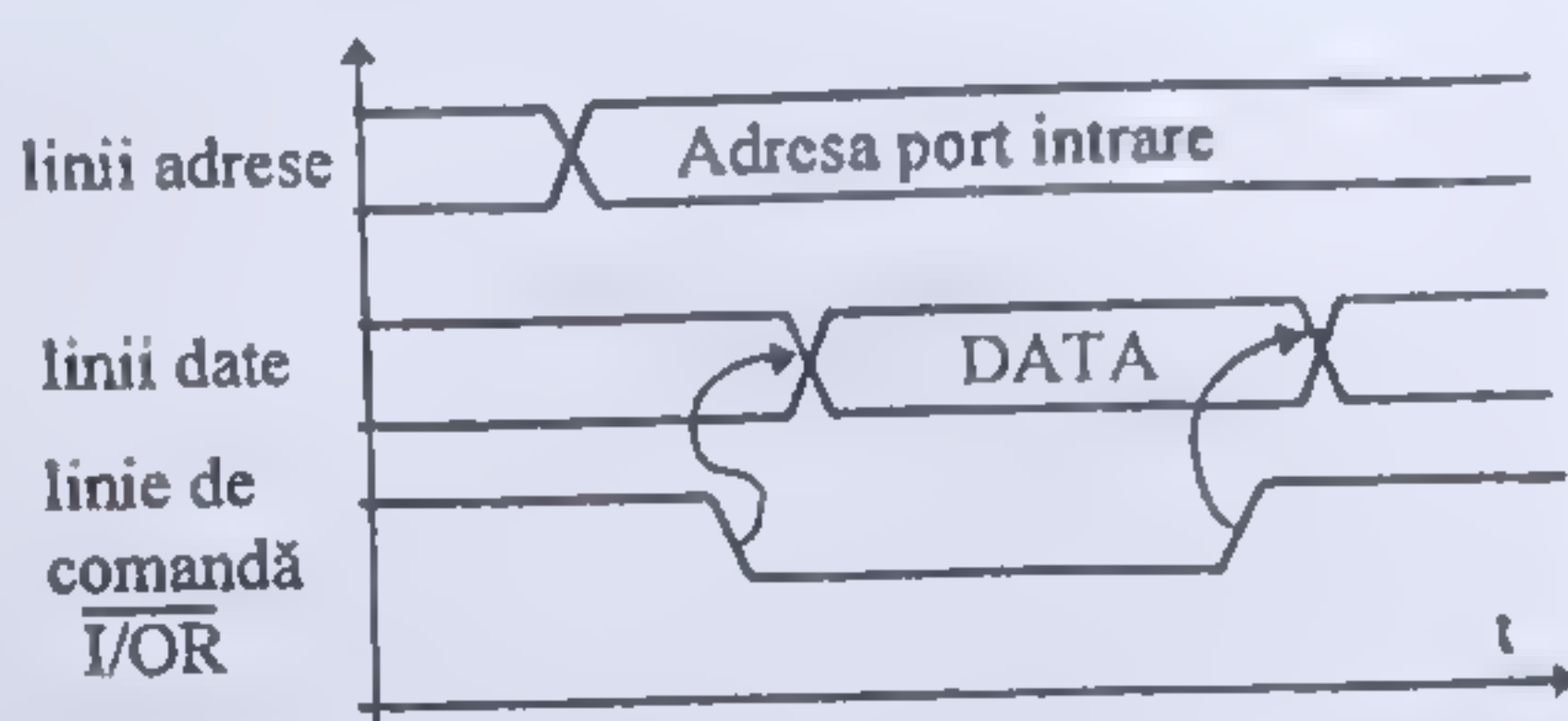


Fig.1.24. Diagrama simplificată la citirea unui port de intrare

Capitolul 1 - Structura sistemelor cu microprocesoare

programabile, microprocesorul trebuie să inițializeze unele registre interne cu cuvinte de comandă, în vederea obținerii regimului de funcționare dorit.

Marea majoritate a porturilor I/E uzuale sunt realizate pentru transferul cuvintelor de un octet și posedă toate facilitățile necesare dialogului cu microprocesoarele.

Pentru porturile de intrare, lungimea cuvântului citit poate fi în relația $l_{CPI} \leq d$ cu magistrala de date, unde l_{CPI} este dimensiunea cuvântului portului de intrare. Însă porturile de ieșire trebuie să aibă capacitatea de a transfera cuvântul furnizat de microprocesor, deci $l_{CPE} = d$, l_{CPE} fiind dimensiunea cuvântului acceptat de un port de ieșire. Evident, dacă $l_{CPE} < d$, este necesar să se conecteze în paralel un număr corespunzător de porturi.

Porturile I/E au încorporate registre care asigură memorarea temporară a informației de la microprocesor sau de la periferic. De asemenea, sunt prevăzute cu o logică de selecție/deselecție care facilitează dialogul cu microprocesorul. Un proiectant de sistem trebuie să aleagă cea mai convenabilă soluție în vederea unei sincronizări perfecte a transferului. Astfel, la dispozitivul de intrare trebuie să se asigure o reținere a informației de la perifericul de intrare, până când aceasta este preluată de către microprocesor. Dispozitivele de ieșire trebuie să memoreze informația de la microprocesor, până la o nouă înscrisere.

Selecția dispozitivelor I/E poate fi, ca și la memorie, *liniară*, cu *decodificare* sau *mixtă*. Pentru exemplificare se va considera dispozitivul I8212 produs de Intel [43], care este un port I/E de 8 biți, de uz general, unidirecțional și neprogramabil (8-bit I/O Port). Circuitul conține un registru tampon format din 8 bistabile de tip D, ale căror ieșiri sunt amplificate ($I_{OL} = 15 \text{ mA}$) și posedă facilitatea 3-stări. De asemenea, conține o logică de comandă și selecție care îi asigură o flexibilitate funcțională apreciabilă. În fig 1.25 este prezentată schema bloc și tabelul de adevăr:

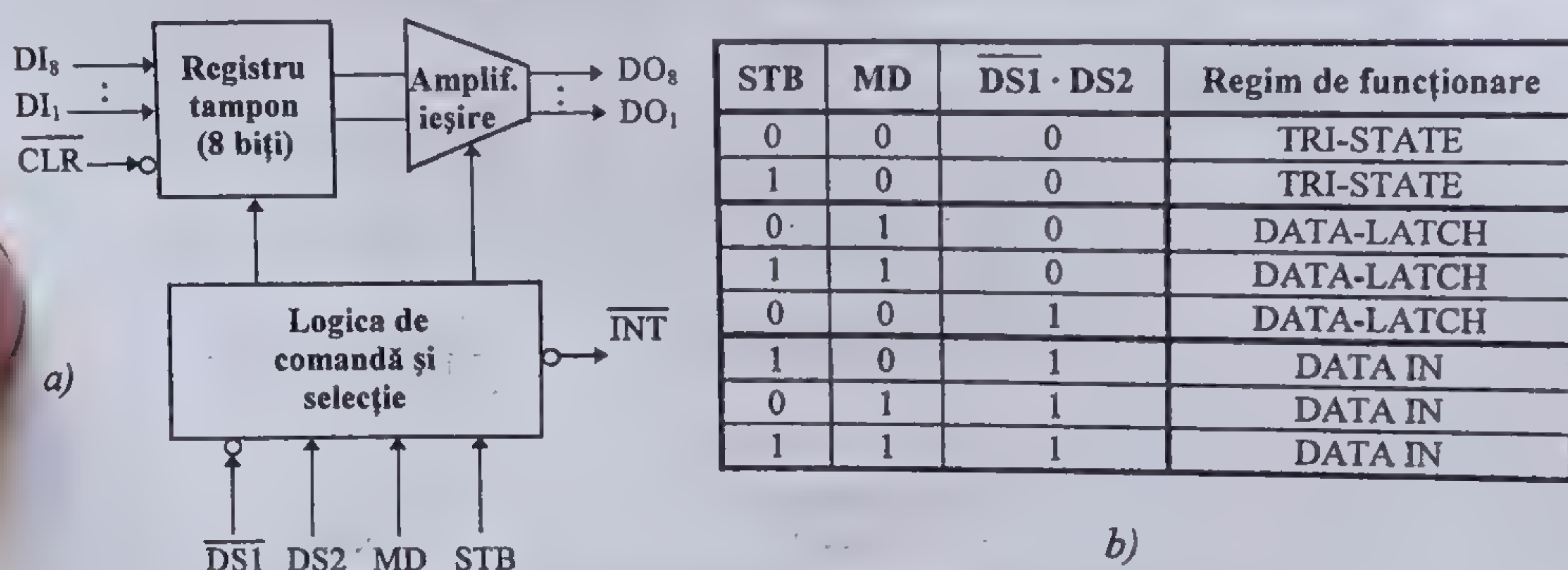


Fig.1.25. Schema bloc și tabelul de adevăr ale portului I8212

Liniile $DI_1 \div DI_8$ (Data Input) consumă numai $250 \mu A$ și permit aplicarea octetului de date la intrările registrului tampon. Acest registru poate fi resetat prin intermediul semnalului \overline{CLR} (Clear).

Funcționarea circuitului este determinată de combinația semnalelor de selecție, $\overline{DS1}$ și $DS2$, de stabilire a modului de lucru, MD (Mode) și de reținere a informației de intrare STB (STroBe), aplicate logicii de comandă și selecție. Prin combinațiile celor 4 semnale se pot obține trei regimuri de funcționare: acceptare date (DATA IN), acceptare și reținere date (DATA LATCH) și înaltă impedanță (TRI-STATE sau HZ). În plus, circuitul 8212 poate lansa o cerere de întrerupere activă pe nivel coborât pe linia \overline{INT} , dacă $\overline{DS1} \cdot DS2 = 1$ (dispozitiv selectat), la apariția unui front descrescător pe intrarea STB.

Din cele prezentate se observă că acest dispozitiv îndeplinește condițiile generale impuse unui port I/E. Pentru conectarea la magistralele sistemului trebuie stabilite regimurile de funcționare corespunzătoare. Astfel, în cazul folosirii circuitului 8212 ca *port de intrare*, este necesar ca liniile sale de ieșire să se afle în stare de înaltă impedanță atât timp cât nu este selectat de către microprocesor. În momentul selectării trebuie să permită transferul datei de la liniile DI pe liniile DO (regimul DATA IN), iar la deselecție liniile DO trebuie să revină în starea de înaltă impedanță. Consultând tabelul din fig.1.25b, rezultă următoarele condiții: $STB=1$, $MD=0$ și $\overline{DS1} \cdot DS2=0$ pentru regimul HZ, respectiv $\overline{DS1} \cdot DS2=1$ pentru DATA IN. Deci, comutarea între cele două moduri de funcționare se poate face controlând corespunzător liniile $\overline{DS1}$ și $DS2$.

În cazul în care circuitul 8212 este utilizat ca *port de ieșire*, este necesar să se asigure controlul trecerii de la regimul de acceptare a datelor de la microprocesor (DATA IN) la regimul de memorare (DATA LATCH), după ce microprocesorul a deselectat portul. Consultând tabelul de adevăr din fig.1.25b, rezultă că cele mai avantajoase condiții sunt: $STB=MD=1$ și comutarea $\overline{DS1} \cdot DS2$ de la valoarea 1 (DATA IN) la valoarea 0 (DATA LATCH).

În fig.1.26 este prezentată schema de conectare a două porturi 8212 - unul de intrare și altul de ieșire - la magistralele unui microprocesor de 8 biți, folosind liniile A_{12} și A_{13} pentru selecție.

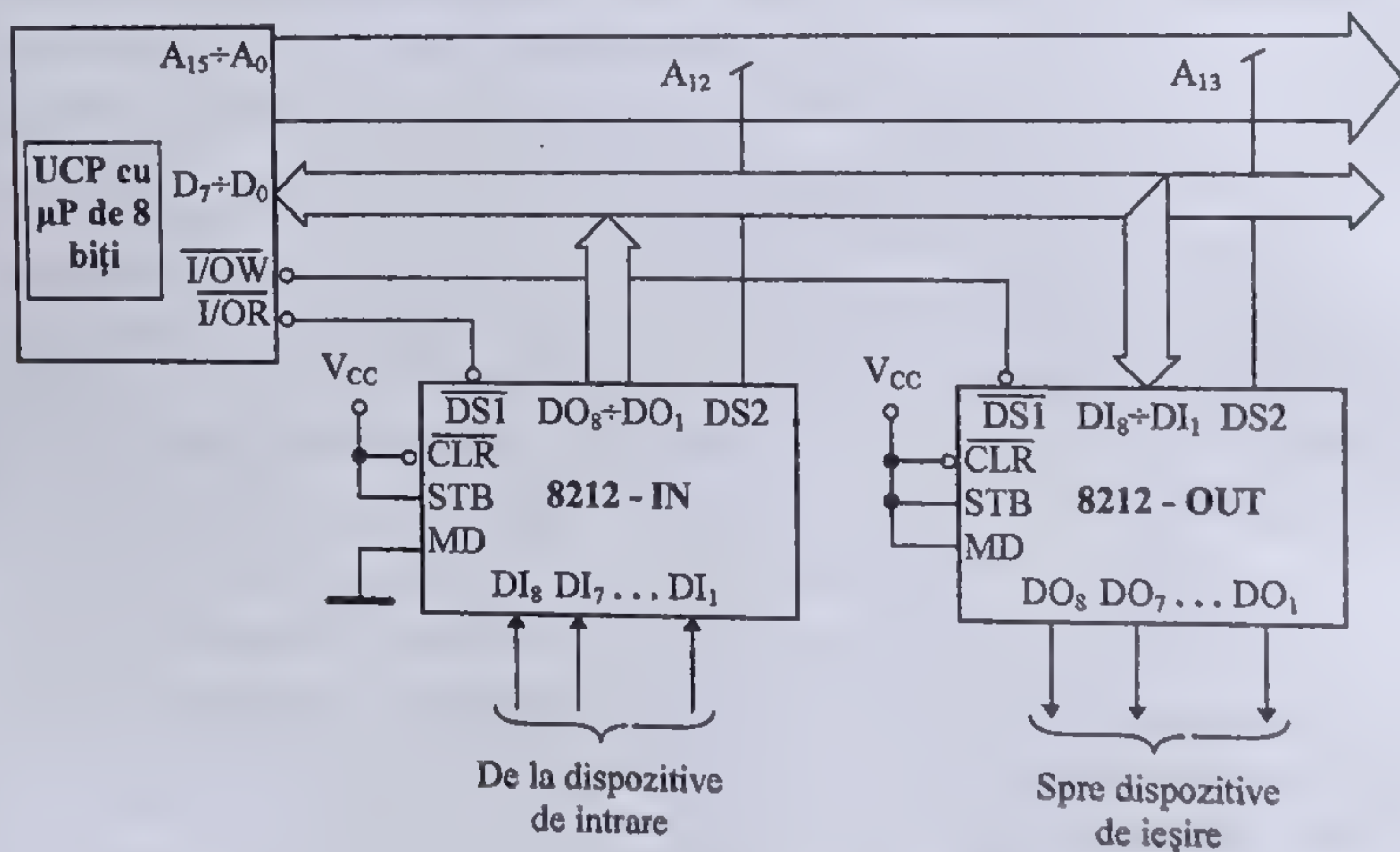


Fig.1.26. Conectarea porturilor I/E la magistrale - selecția liniară

În fig.1.27 este ilustrat cazul selecției cu decodificare (un decodificator 2/4) folosind o variantă simplificată de control prin linia $\overline{DS1}$, linia $DS2$ fiind conectată continuu la 1 logic și fără utilizarea liniilor $\overline{I/OR}$ și $\overline{I/OW}$. Acest lucru este posibil în cazul porturilor unidirecționale, dacă se respectă procedurile de citire, respectiv de scriere, precum și adresarea corespunzătoare a porturilor de intrare (8212 - IN) și de ieșire (8212 - OUT).

Nu același lucru se întâmplă în cazul porturilor bidirecționale, când este necesar ca microprocesorul să specifice tipul de operație în mod explicit. Fără a intra în detalii, s-a considerat pentru exemplificare, în fig.1.27, dispozitivul Intel 8255, care este o interfață paralelă programabilă, bidirecțională. Acesta dispune de două linii de comandă al tipului de tranfer (scriere/citire - \overline{WR} și \overline{RD}), o linie de selecție (\overline{CS}) și două linii de adresare (A_1 , A_0).

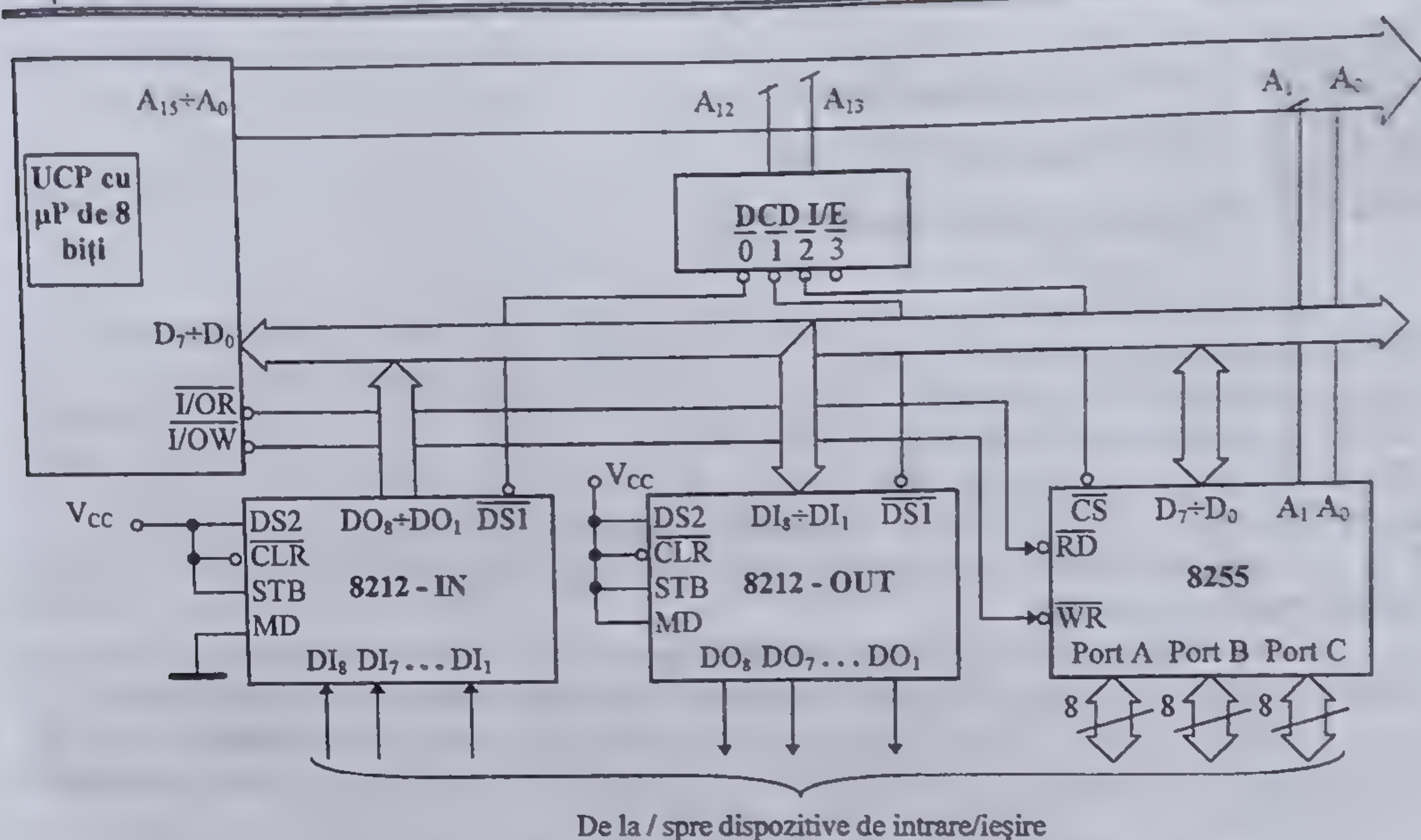


Fig.1.27. Conectarea porturilor I/E la magistrale - selecția cu decodificare

decodificate intern, pentru selecția unuia din cele trei porturi I/E ale dispozitivului (A, B și C) sau a registrului de programare a modului de lucru.

2

Interfațarea sistemelor cu microprocesoare

O problemă importantă care trebuie rezolvată la realizarea sistemelor cu microprocesoare pentru conducerea proceselor este cuplarea, prin intermediul unor interfețe adecvate, a perifericelor necesare dialogului cu operatorul și conducerii proceselor fizice. Dintre perifericele cel mai des folosite se pot menționa: tastaturi - pentru introducerea comenzilor, dispozitive de afișare - pentru afișarea informațiilor din proces, unități de memorie pe suport magnetic - pentru păstrarea bazei de date a sistemului, precum și sistemul de interfață cu procesul condus. Fizic, astfel de periferice pot fi de la simple comutatoare, butoane și LED-uri, până la convertoare A/N - N/A, tastaturi și sisteme complexe de afișare.

De asemenea, implementarea conceptului de conducere distribuită se materializează prin sisteme cu microprocesoare distribuite fizic, dar conectate între ele prin legături informaționale.

Ambele situații necesită existența unor interfețe de comunicație specializate, care să permită transferul datelor în format paralel sau serial, prin utilizarea tehnicilor I/E ce urmează a fi prezentate în §2.3.

Un rol important în implementarea unora din aceste tehnici I/E îl are introducerea timpului ca variabilă în prelucrarea informației, în special în aplicațiile care necesită o planificare și o execuție în timp real a activităților de conducere, așa cum se va arăta în §2.4.

2.1. Interfațarea paralelă

Deoarece numărul semnalelor de la/spre periferice este relativ mare, iar tratarea este diferită, în funcție de complexitatea și importanța lor în cadrul sistemului, pentru interfațarea cu perifericele a sistemelor cu microprocesoare se utilizează, de obicei, *porturi I/E paralele programabile*.

Metode pentru transferul paralel al datelor

Pentru o interfațare corectă cu diferite periferice este necesar să se cunoască principalele modalități de transfer a datelor în format paralel.

a) **Intrare/ieșire simplă.** În acest caz, datele pot fi transferate de la sau către microprocesor, prin intermediul porturilor I/E, în orice moment, dar numai la inițiativa și sub controlul unității centrale. Deci, la acest tip de transfer, perifericul joacă un rol pasiv, iar interfața paralelă are doar rolul de adaptare a caracteristicilor electrice ale liniilor de date dintre periferic și magistrala sistemului (fig.2.1a).

b) **Intrare/ieșire strobată.** În multe aplicații, datele de la un periferic sunt valide numai la anumite momente de timp, când sunt acompaniate de un *semnal de strob* (fig.2.1b). La intrare,

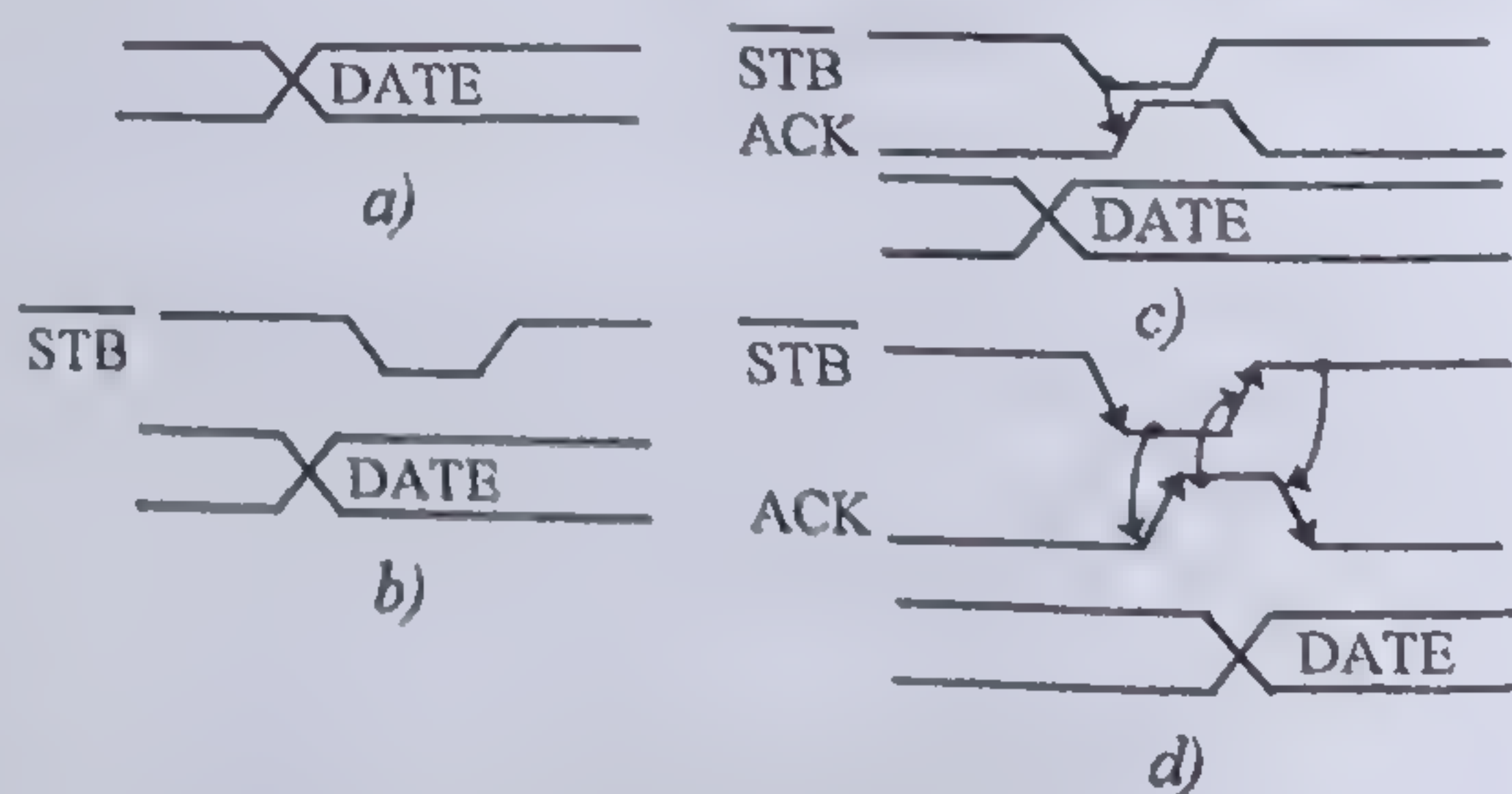


Fig.2.1. Tipuri de transfer paralel de date:
a) I/E simplă; b) I/E strobată; c) I/E cu confirmare simplă; d) I/E cu confirmare dublă

Capitolul 2 - Interfațarea sistemelor cu microprocesoare

semnalul \overline{STB} este activat de periferic pentru a marca momentul în care data este validă la intrarea portului și declanșează o procedură de întrerupere sau de interogare. La ieșire, microprocesorul trebuie să transmită atât semnalul de strob, cât și data; semnalul \overline{STB} va fi folosit de periferic pentru a reține data furnizată de microprocesor prin portul de ieșire.

c) **Intrare/ieșire cu confirmare simplă (single-handshake).** Pentru rate ridicate de transfer, metoda precedentă nu poate fi folosită, deoarece nu există nici o confirmare că datele au fost recepționate și pot fi transmise altele noi. Cu alte cuvinte, transmitătorul ar putea ajunge în situația de a transmite date mai rapid decât pot fi acceptate de receptor. Rezolvarea acestei probleme se poate face prin folosirea unei scheme de *transfer cu confirmare (handshake)*.

Spre exemplu, un periferic transmite datele paralel, către microprocesor, prin intermediul unui port de intrare (fig.2.2).

Existența unei date valide la ieșirea perifericului este anunțată prin activarea semnalului \overline{STB} . Microprocesorul detectează, prin intermediul portului de intrare, activarea acestui semnal (de exemplu folosind sistemul de întreruperi) și citește datele. Apoi, portul de intrare transmite către periferic un semnal de recunoaștere (ACKnowledge), confirmând recepția și, implicit, acceptarea unei noi date. De obicei, interconținerea dintre semnalele \overline{STB} și ACK se realizează, la transferul cu confirmare simplă, ca în fig.2.1c.

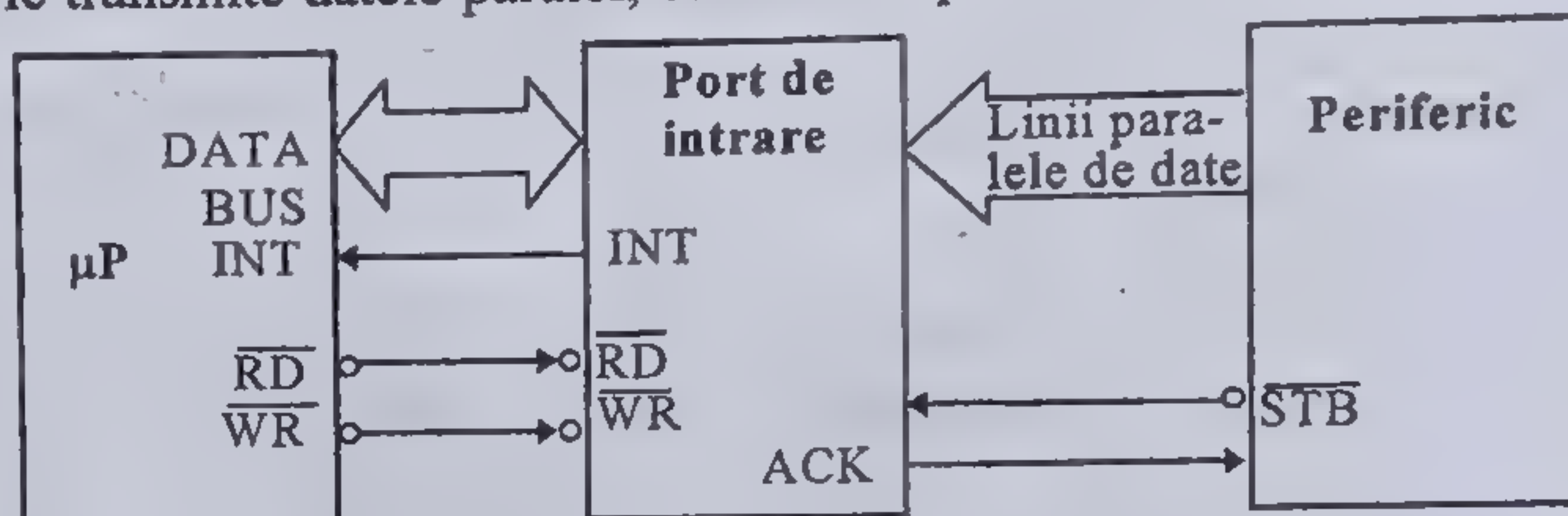


Fig.2.2. Schema de transfer cu confirmare a datelor de intrare

d) **Intrare/ieșire cu dublă confirmare (double-handshake).** În cazul în care se cere o mai bună coordonare a transferului de date decât cea oferită de confirmarea simplă, se apelează la procedura prezentată în fig.2.1d. Mai întâi, transmitătorul lansează o procedură de pregătire a receptorului: când semnalul \overline{STB} este activat, receptorul trebuie să-și confirme disponibilitatea de a efectua transferul, prin activarea semnalului ACK. Dacă această fază se încheie cu succes (ACK=1), transmitătorul depune informația pe liniile de date și apoi dezactivează linia \overline{STB} , anunțând astfel validitatea datei. Receptorul citește data și confirmă preluarea ei prin dezactivarea liniei ACK, dacă și $\overline{STB}=1$. Se observă că transferul de date implică, în acest caz, o dublă confirmare a fiecărei faze, de unde provine și denumirea metodei. La intrare, conexiunile sunt identice cu cele din fig.2.2. Pentru o ieșire de date (transmisie de la microprocesor la periferic), formele de undă sunt aceleași, dar microprocesorul este cel care transmite data și semnalul \overline{STB} , perifericul furnizând semnalul de confirmare ACK.

Implementarea tehnicilor cu confirmare se realizează prin intermediul interfețelor paralele programabile, care dispun de toate semnalele de strobare și de confirmare necesare. Un astfel de dispozitiv trebuie să conțină cel puțin un registru tampon la intrare și un registru tampon la ieșire, care să mențină data până la stabilirea legăturii dintre microprocesor și periferic. În general, pentru comunicația paralelă cu un periferic se utilizează mai mult de un port bidirecțional, de obicei două sau trei porturi de 8 biți, multiplexate.

În fig.2.3 este prezentată schema bloc principală a unei interfețe paralele cu două porturi de ieșire: Port 1 și Port 2. Fiecare port are propriul său bloc de comandă, un registru tampon și un registru direcțional, care va fi încărcat cu o informație ce stabilește sensul de transmitere a datelor. De asemenea, interfața conține un registru al modului de lucru, care permite microprocesorului să specifice tipul de transfer pentru fiecare port în parte (cu sau fără confirmare), precum și sensul de

transfer (intrare sau ieșire). Pentru fiecare port sunt prevăzute semnale de comandă, necesare implementării transferului cu confirmare (\overline{STB} , \overline{ACK} etc.).

Întregul circuit este prevăzut cu un bloc de logică de selecție (\overline{CS} sau \overline{RS} - Register Select), controlat prin magistralele de adrese și de comandă, care asigură dialogul cu microprocesorul (\overline{RD} , \overline{WR} și linii de întrerupere \overline{INT} de la fiecare port I/E).

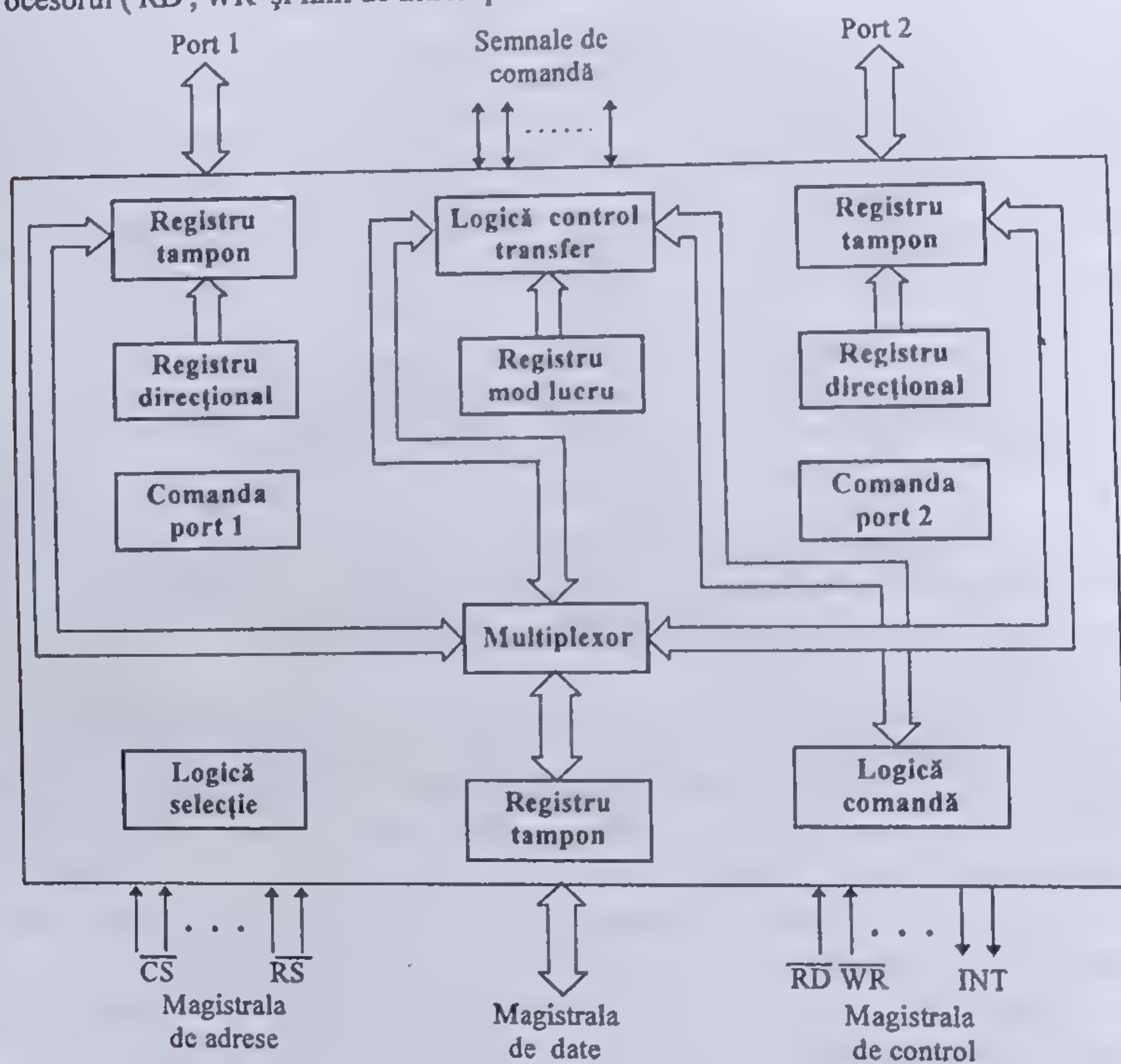


Fig.2.3. Schema de principiu a unei interfețe paralele I/E programabile

Principalele firme producătoare de microprocesoare furnizează și interfețe paralele I/E programabile, cele mai cunoscute fiind: PPI 8255A (Programmable Peripheral Interface) - Intel (cu 3 porturi I/E de 8 biți); PIO-Z80 (Programmable Input/Output) - Zilog (cu 2 porturi de 8 biți); PIA-6820 (Peripheral Interface Adapter) - Motorola (cu 2 porturi de 8 biți).

2.2. Interfațarea serială

Aceasta este realizată cu ajutorul unor circuite programabile specializate, de tip transmițător/receptor serial asincron - UART (Universal Asynchronous Receiver/Transmitter, spre ex.: TMS6011, ACIA-6850, INS8250, INS16450, INS16500) sau și de tip sincron - USART (Universal Synchronous/Asynchronous Receiver/Transmitter, de ex.: 18251, 18251A, 2651, SIO-Z80). Schema bloc generală a unui astfel de dispozitiv se prezintă în fig.2.4. Blocurile componente comunică între ele printr-o magistrală internă, conectată printr-un tampon bidirecțional la

Capitolul 2 - Interfațarea sistemelor cu microprocesoare

magistrala de date externă. Microprocesorul deține rolul de master în dialogul cu interfața serială, furnizând semnalele de comandă necesare sincronizării schimbului de date în format paralel. Momentele în care este necesar acest schimb de date sunt marcate de activarea semnalelor de întrerupere la transmisie și la recepție, furnizate microprocesorului de către interfața serială.

De cealaltă parte a interfeței, cea a liniei de comunicație serială, este prevăzut un bloc de control, necesar pentru sincronizarea transferului de date cu modemul (un dispozitiv care asigură interfața cu mediul fizic de comunicație), precum și un canal duplex, transmițător/receptor, a cărui structură internă se prezintă în fig.2.5.

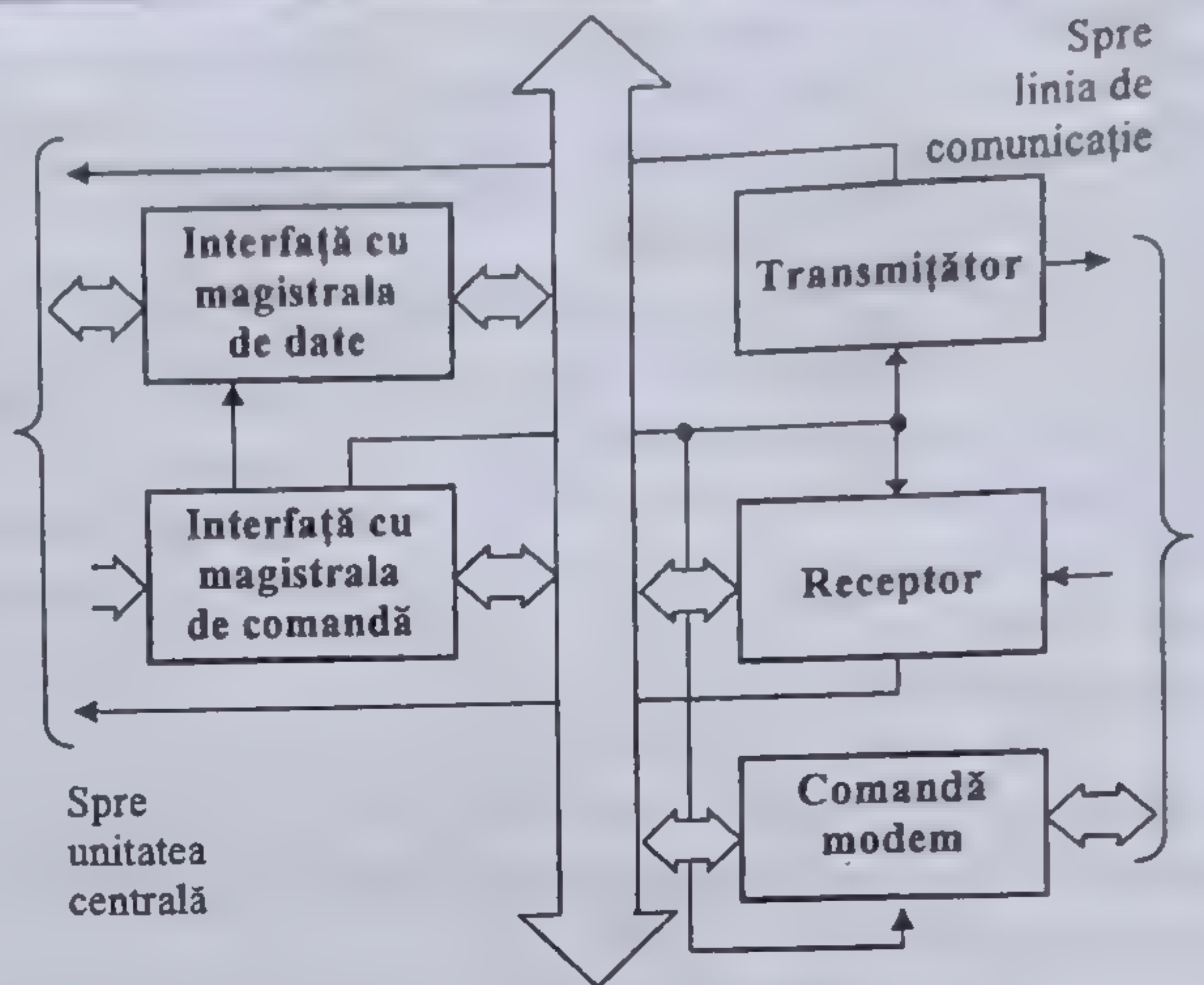


Fig.2.4. Structura generală a unei interfețe seriale

Transmițătorul conține un registru tampon de transmisie, în care microprocesorul înscrie data la activarea semnalelor corespunzătoare de selecție și de înscriere ($C/\overline{D}=0$ și $\overline{WR}=0$). Data este transferată automat într-un registru de deplasare paralel/serie, imediat ce acesta a terminat transmisia datei anterioare. Ieșirea serie a acestui registru constituie linia de transmisie serială a datelor și este notată în fig.2.5 cu TxD. În funcție de modul de lucru, asincron sau sincron, logica

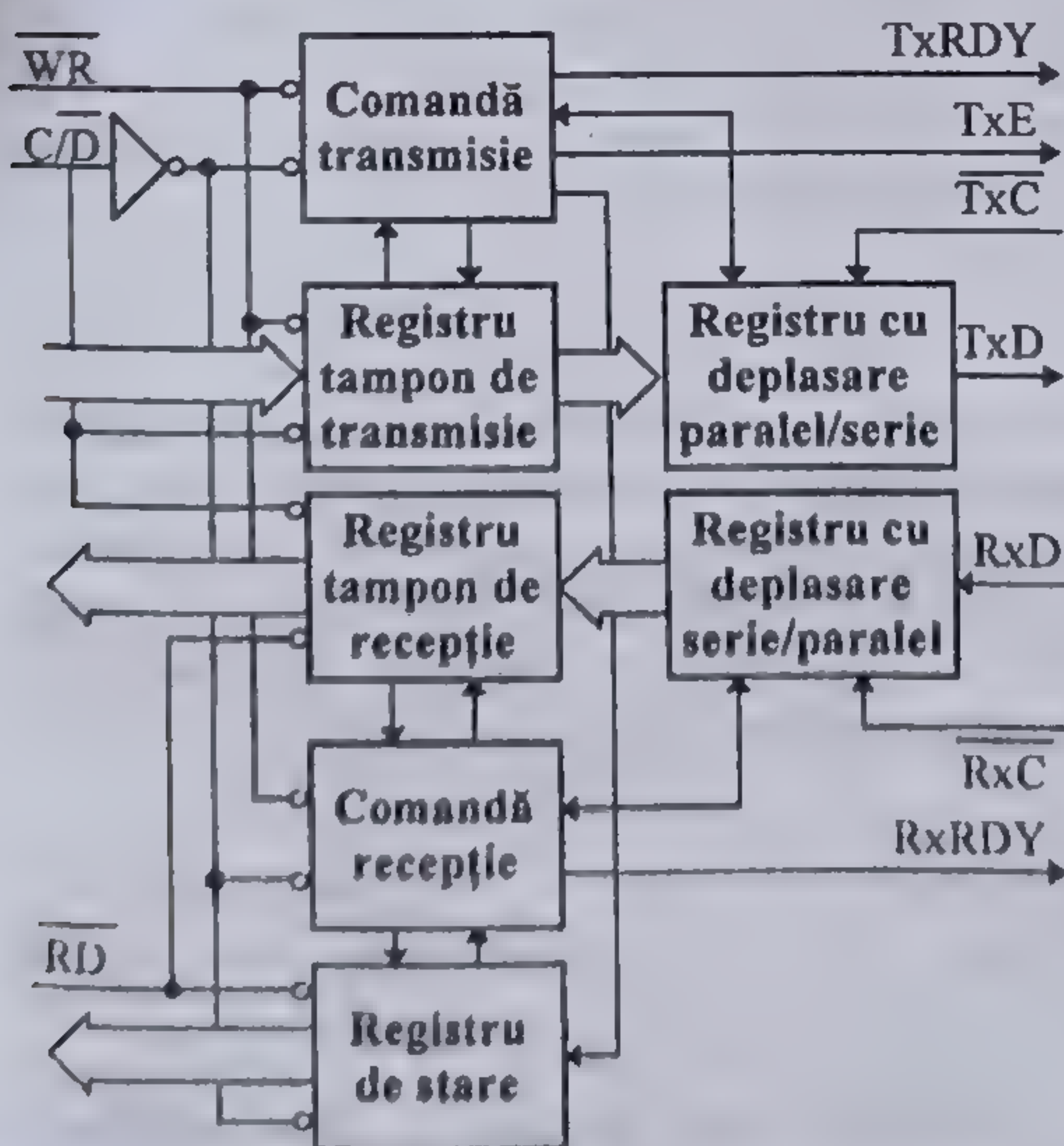


Fig.2.5. Structura unui canal transmițător/receptor

de comandă a transmisiei adaugă la biții de date, biți de control și de sincronizare. Semnalul de tact care determină deplasarea biților cu o poziție în registrul paralel/serie poate fi \overline{TxC} sau un semnal obținut prin divizarea acestuia (de obicei cu 16 sau cu 64). Semnalul \overline{TxC} poate fi generat din interiorul sistemului, prin divizarea semnalului de tact al unității centrale, sau poate fi primit din exterior (de la modem).

Programarea transmițătorului constă din înscrierea de către UCP (cu $C/\overline{D}=1$ și $\overline{WR}=0$), în registrele logicii de comandă, a unor cuvinte de control, prin care se stabilesc caracteristicile funcționale și modul de operare al circuitului. Unitatea centrală poate primi informații despre starea transmițătorului prin liniile TxRDY și TxRDY. Astfel, TxRDY devine activ la vederea registrului tampon de transmisie, când

UC poate înscrie un nou caracter. TxE este activat atunci când și registrul cu deplasare paralel/serie este gol, deci efectiv nu mai sunt caractere de transmis. Aceste informații ajung la UC prin întreruperi generate de liniile TxRDY, respectiv TxE, sau prin citirea (cu $C/\overline{D}=1$ și $\overline{RD}=0$) cuvântului de stare a canalului.

Receptorul conține un registru de deplasare serie/paralel, în care sunt introduși biții caracterelor, recepționați în mod serial, pe linia RxD. Citirea liniei RxD și deplasarea cu un rang a biților în registrul serie/paralel au loc la momente bine stabilite, marcate de semnalul de tact \overline{RxC} sau de un semnal obținut, la fel ca în cazul transmițătorului, prin divizarea cu 16 sau cu 64 a lui \overline{RxC} . De îndată ce în registrul cu deplasare s-a recepționat numărul de biți necesar pentru formarea unui cuvânt de date, acesta este transferat în format paralel în registrul tampon de recepție, după care registrul cu deplasare poate să preia, în aceeași manieră, biții care urmează.

Unitatea centrală poate afla că în registrul tampon de recepție există o dată nepreluată, fie printr-o cerere de întrerupere furnizată pe linia RxRDY, fie prin citirea cuvântului de stare a canalului. Caracterul recepționat este citit de microprocesor la activarea semnalelor de selecție și de citire date ($C/\overline{D}=0$ și $\overline{RD}=0$).

Programarea receptorului se face prin înscrierea, de către microprocesor, a cuvintelor de control și de comandă în registrele logicii de comandă a recepției, corespunzător modului de lucru și parametrilor doriți. Cuvântul de stare al canalului conține, în trei indicatori de eroare și informații legate de modul în care a decurs recepția ultimului cuvânt de date.

Operațiile de citire și de refacere a formatului datelor la recepție impun existența unui anumit grad de coordonare între transmițător și receptor. Schemele lor interne trebuie sincronizate la începutul transmisiei fiecărui bloc de date, iar sincronizarea trebuie menținută în interiorul blocului de date, la nivel de bit, pe baza semnalelor de tact ale transmițătorului și receptorului.

Pentru o legătură în banda de bază, fără modem (null modem), ecartul de frecvență dintre cele două semnale de tact trebuie să fie cât mai mic, astfel încât să poată fi păstrat sincronismul la nivel de bit, pentru un număr cât mai mare de biți ai blocului de date. Spre deosebire de comunicația sincronă, când sincronismul stabilit la începutul blocului de date trebuie păstrat pe durata transferului unui număr mare de biți, în cazul comunicației asincrone blocul de date este format dintr-un singur caracter, de unde și condițiile mai puțin severe în ceea ce privește ecartul de frecvență dintre semnalele de tact ale transmițătorului și receptorului.

2.3. Tehnici de intrare-ieșire

Modalitățile de sincronizare între microprocesor și dispozitivele I/E poartă numele de *tehnici de intrare-ieșire*. Pentru comanda operativă, în timp real a sistemului, aceste tehnici sunt de mare importanță, întrucât de ele depinde eficiența transferului de informație intrare-ieșire. Atât din acest punct de vedere, cât și pentru asigurarea integrității informației transferate, este necesară sincronizarea funcționării microprocesorului cu evenimentele externe și tratarea lor corespunzătoare. În acest scop s-au cristalizat următoarele tehnici de intrare-ieșire:

- a) transferul programat I/E;
- b) transferul I/E cu utilizarea întreruperilor;
- c) transferul I/E prin acces direct la memorie.

2.3.1. Transferul programat I/E

Acesta presupune rularea periodică, de către microprocesor, a unui *program de interogare* a dispozitivelor I/E conectate la sistem, în vederea detectării momentului în care acestea solicită sau sunt pregătite pentru un transfer I/E.

Capitolul 2 - Interfațarea sistemelor cu microprocesoare

Această tehnică, numită și *de interogare a dispozitivelor* (devices polling), presupune că fiecare dispozitiv I/E conține un *registru de stare*, care poate fi citit în orice moment de către microprocesor. Cuvântul citit poartă numele de *cuvânt de stare* și reflectă situația în care se află dispozitivul I/E în acel moment. Practic, un dispozitiv extern pregătit pentru transfer setează un bit în registrul său de stare, prin care solicită microprocesorului un transfer I/E. Dispozitivele I/E sunt interogate pe rând de către microprocesor, prin citirea registrului de stare și testarea valorii bitului corespunzător din cuvântul citit. Dacă microprocesorul detectează o solicitare, lansează un program care realizează transferul I/E.

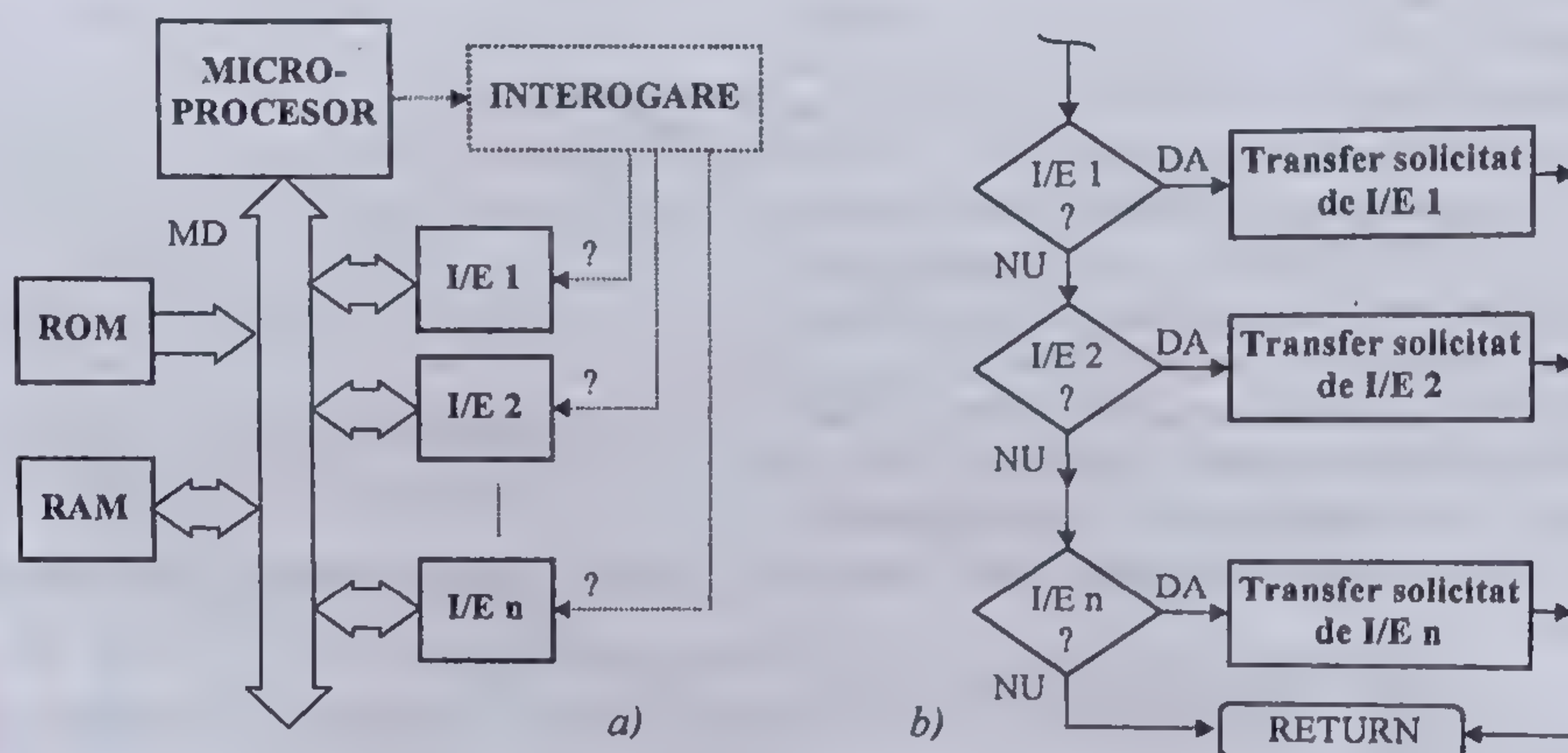


Fig.2.6. Schema simplificată a transferului programat (cu interogare)

În fig.2.6 este prezentată schema simplificată a transferului programat. Prin linii întrerupte, în fig.2.6a s-a sugerat procedura de interogare, iar în fig.2.6b - schema logică corespunzătoare. Avantajul procedurii constă în simplitatea sa. Dezavantajele rezidă în ocuparea UCP cu rularea repetată a procedurii de interogare, în timpul mare de răspuns la solicitări, precum și în faptul că metoda impune o ierarhie fixă în tratare, ceea ce poate conduce la defavorizarea dispozitivelor din coada ierarhiei. Unele variante ale acestei metode pot să elimine ultimul dezavantaj, dar printr-o complicare a procedurii de tratare, care crește și mai mult timpul de răspuns și gradul de ocupare al procesorului.

2.3.2. Transferul intrare-ieșire cu utilizarea întreruperilor

Acesta implică utilizarea regimului de întreruperi al microprocesorului. Dacă există mai multe dispozitive care pot solicita în acest mod un transfer I/E, este necesar un sistem de întreruperi (vezi fig.1.1)

În fig.2.7 este prezentată schema generală a transferului I/E prin întreruperi. Un *controler de întreruperi* gestionează solicitările de la dispozitivele I/E: primește cererile pe liniile IRQ1, ..., IRQn, alege dispozitivul cel mai prioritar în cazul apariției mai multor cereri simultane și anunță microprocesorul de necesitatea efectuării unui transfer I/E. Microprocesorul își întrerupe programul curent, salvează starea programului în stivă și trece la tratarea transferului pentru dispozitivul I/E indicat de controler. În acest mod se realizează, prin hardware, o sincronizare cu funcționarea UCP și se asigură un răspuns în timp real. După terminarea transferului, microprocesorul reface starea programului și reîncepe execuția din punctul în care aceasta a fost întreruptă.

La unele microprocesoare (spre exemplu Z80), tratarea întreruperilor pe mai multe niveluri de prioritate se realizează prin intermediul unei logici interne specifice, integrată în structura dispozitivelor I/E ale familiei, nemaifiind necesar un controler extern specializat.

În ambele cazuri, după acceptarea întreruperii de către microprocesor, controlerul de întreruperi sau dispozitivul I/E solicitant furnizează un vector de întrerupere, motiv pentru care această tehnică poartă numele de *întreruperi vectorizate* (vectored interrupts).

Vectorul de întrerupere poate fi chiar *adresa efectivă* a rutinei de tratare a întreruperii sau doar un cod de identificare a dispozitivului solicitant (adresă de dispozitiv), de un octet. În ultimul caz, pe baza vectorului de întrerupere, microprocesorul determină o adresă de memorie, numită *adresă indirectă* de tratare. La această adresă se află stocată *adresa efectivă* a rutinei de tratare. Acest mecanism are două avantaje esențiale:

- nu solicită o informație complexă din partea dispozitivului I/E;
- conferă o relativă independență a dispunerii rutinelor de deservire a întreruperilor în spațiul de memorie.

Cele arătate mai sus au impus tehnica întreruperilor vectorizate în sistemele cu microprocesoare, motiv pentru care i se va acorda un spațiu mai amplu la studierea principalelor tipuri de microprocesoare, cu aprofundarea modului specific de rezolvare.

2.3.3. Transferul intrare-ieșire prin acces direct la memorie

Se realizează prin trecerea microprocesorului în regimul de cedare de magistrale (vezi §1.1.1.3). Și în acest caz este necesar un dispozitiv specializat, de tip *controler DMA*, care gestionează solicitările de acces direct la memorie ale altor dispozitive I/E, arbitrează solicitările simultane pe baza priorităților de servire și lansează o cerere, BRQ, către microprocesorul UCP (fig.1.1 și fig.2.8.).

Microprocesorul răspunde la solicitare (prioritar față de o cerere de întrerupere) prin activarea semnalului BACK și își suspendă activitatea. Controlerul DMA poate fi extern (cel mai adesea) sau intern, inclus în structura microprocesorului (de exemplu: CDP 1802, al firmei RCA).

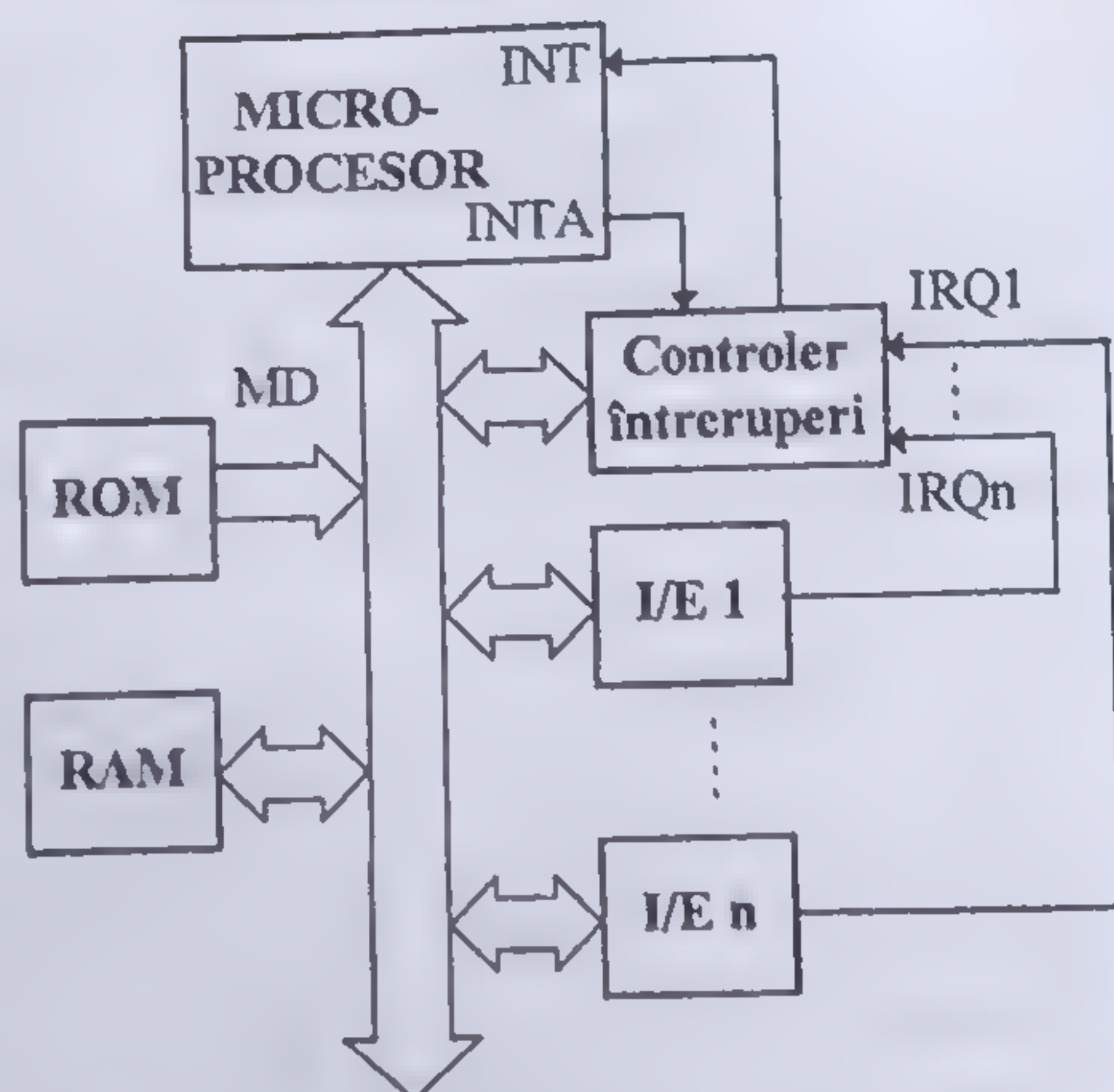


Fig.2.7. Schema simplificată a transferului I/E cu utilizarea întreruperilor

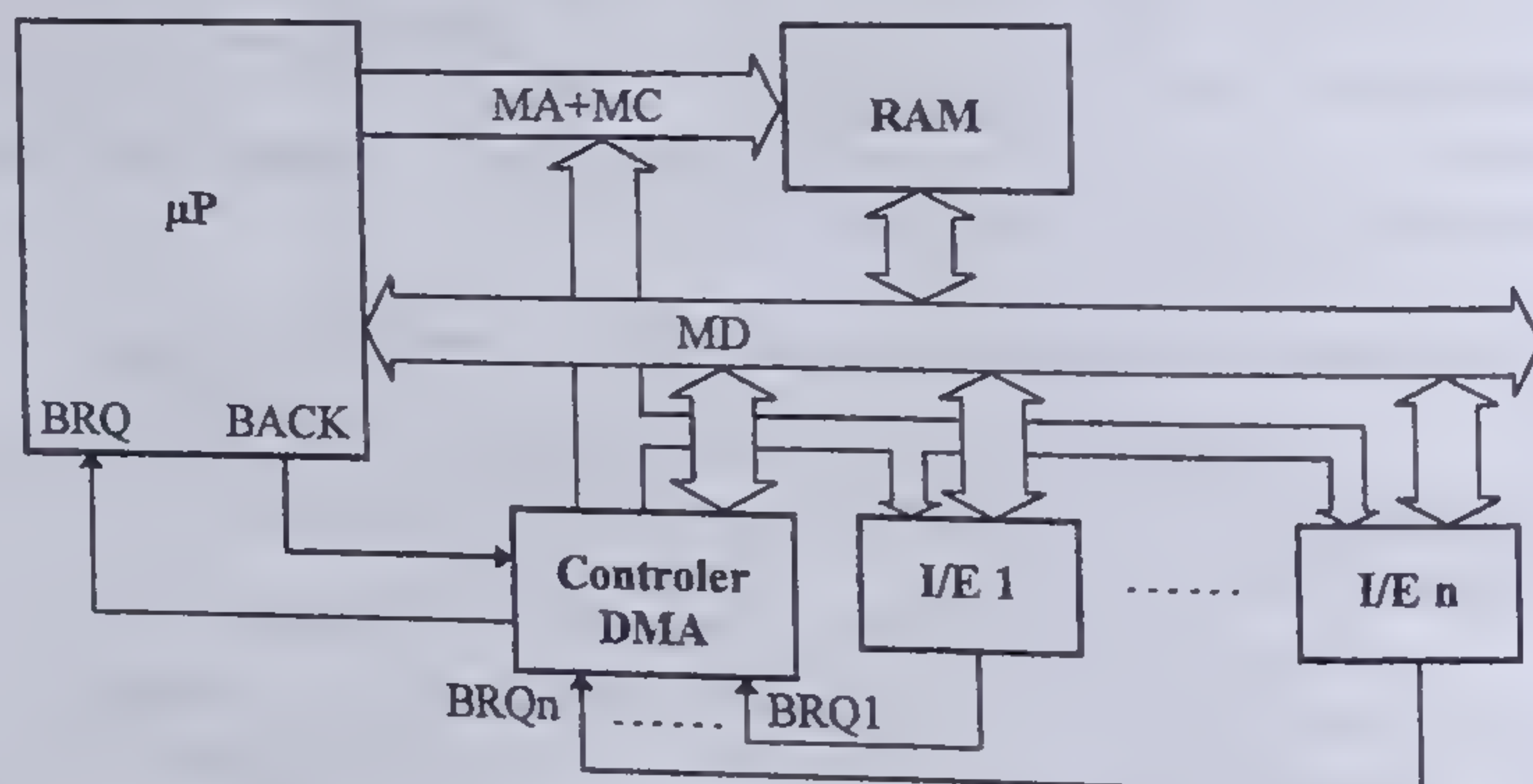


Fig.2.8. Schema transferului I/E prin acces direct la memorie

Capitolul 2 - Interfațarea sistemelor cu microprocesoare

În momentul în care devine coordonator de sistem, controlerul generează adresele și semnalele de control pentru memoria RAM și pentru dispozitivul I/E tratat prin tehnica DMA. Transferul de informație are loc pe magistrala de date, direct între memorie și dispozitivul I/E, fără intervenția microprocesorului. Din acest motiv, tehnica DMA asigură cea mai ridicată rată de transfer I/E, dar necesită prezența unui controler DMA, care este un dispozitiv de o complexitate apropiată de cea a microprocesorului.

2.4. Tehnici de introducere a timpului ca variabilă în prelucrarea informației

Evoluția în timp real a proceselor fizice face ca variabila "timp" să fie esențială în sistemele de conducere automată, fiind asociată în special cu următoarele cerințe:

- generarea de semnale care să marcheze intervale de timp de durată prestabilită, variabilă sau fixată;
- măsurarea timpului scurs între două evenimente;
- înregistrarea timpului real (astronomic).

Primul caz se referă la colectarea unor mărimi din proces la anumite intervale de timp sau la aplicarea unor mărimi de comandă pe o anumită durată. Astfel, achiziția datelor analogice din proces cu o anumită perioadă de eșantionare, rularea periodică a unor programe de autotest, editarea protocoalelor orare, pe schimb etc. sunt evenimente care se produc la momente bine definite de timp. În aceste condiții, devine avantajoasă folosirea unor "ceasuri" (timers) care declanșează cereri de întrerupere la intervale de timp date. Cererile de întrerupere sunt arbitrate de controlerul de întreruperi al sistemului, așa cum se arată în fig.2.9.

Tot în această categorie se încadrează și generarea unor semnale periodice cu frecvența reglabilă (programabilă) pentru sincronizarea funcționării unor circuite de interfață din sistem (spre exemplu - generatoare de semnale de tact pentru interfețele de comunicație serială).

A doua situație este întâlnită în cazul măsurării intervalelor de timp dintre două evenimente. Astfel de necesități pot apare în cazul aproximării derivatelor unor mărimi fizice în raport cu timpul, cum ar fi calcularea unor viteze, debite, productivități etc. Se pot folosi în acest caz două proceduri:

- cele două evenimente citesc succesiv un ceas, intervalul de timp fiind calculat prin diferență;
- primul eveniment pornește ceasul, iar cel de-al doilea îl oprește, după care îi citește valoarea.

Prima procedură este mai simplă de implementat, același ceas fizic putând fi utilizat simultan de mai multe perechi de evenimente (fig.2.10a); are însă dezavantajul scăderii preciziei de măsurare odată cu creșterea numărului de perechi de evenimente care îl utilizează. A doua soluție (fig.2.10b) este mai scumpă

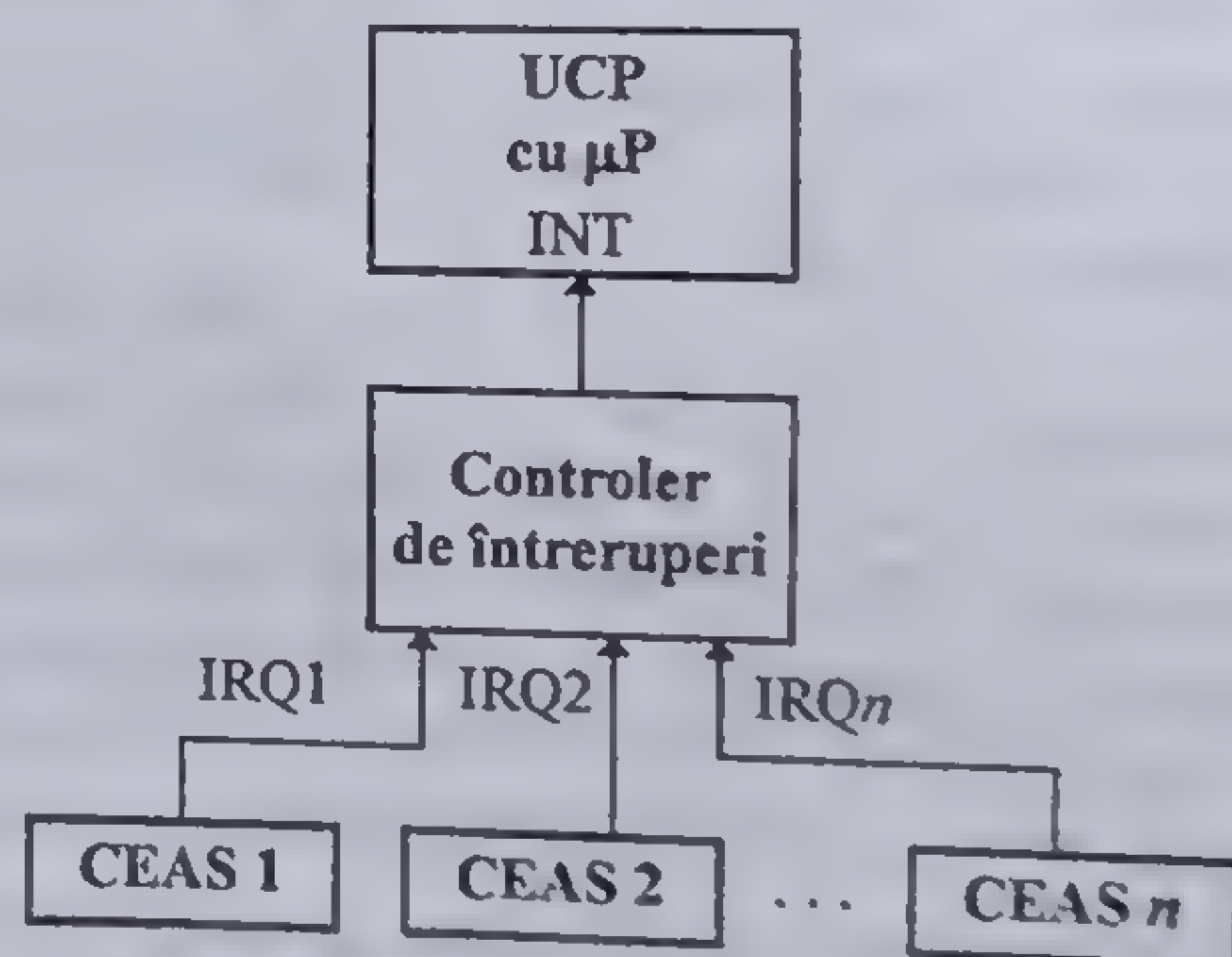


Fig.2.9. Generarea de întreruperi la momente de timp prestabilite

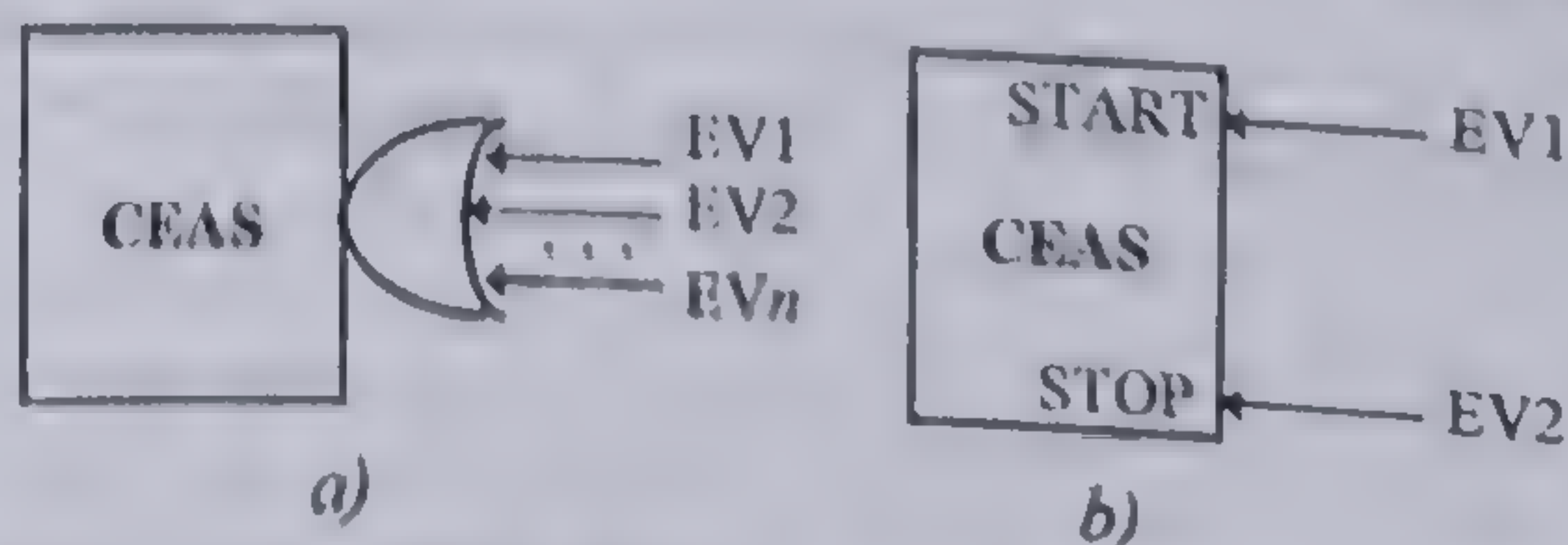


Fig.2.10. Măsurarea timpului scurs între două evenimente

(necesită existența a câte unui ceas pentru fiecare pereche de evenimente), dar asigură o precizie ridicată în măsurarea intervalelor de timp dintre evenimente.

Tot în această clasă de facilități se încadrează un caz special, mult folosit în conducerea în timp real și anume *ceasul de gardă* (Watch Dog Timer - WDT). Acesta se utilizează la supravegherea duratei de desfășurare a unei activități periodice, indicând faptul că s-a depășit valoarea prestabilită.

Spre exemplu, semnalizarea depășirii duratei maxime estimate pentru execuția unui program poate evidenția o funcționare eronată a sistemului (v.fig.2.11). La intrarea în program ($y=0$) se activează semnalul x ($x=1$), ceea ce determină trecerea ieșirii \bar{Q} a mono-

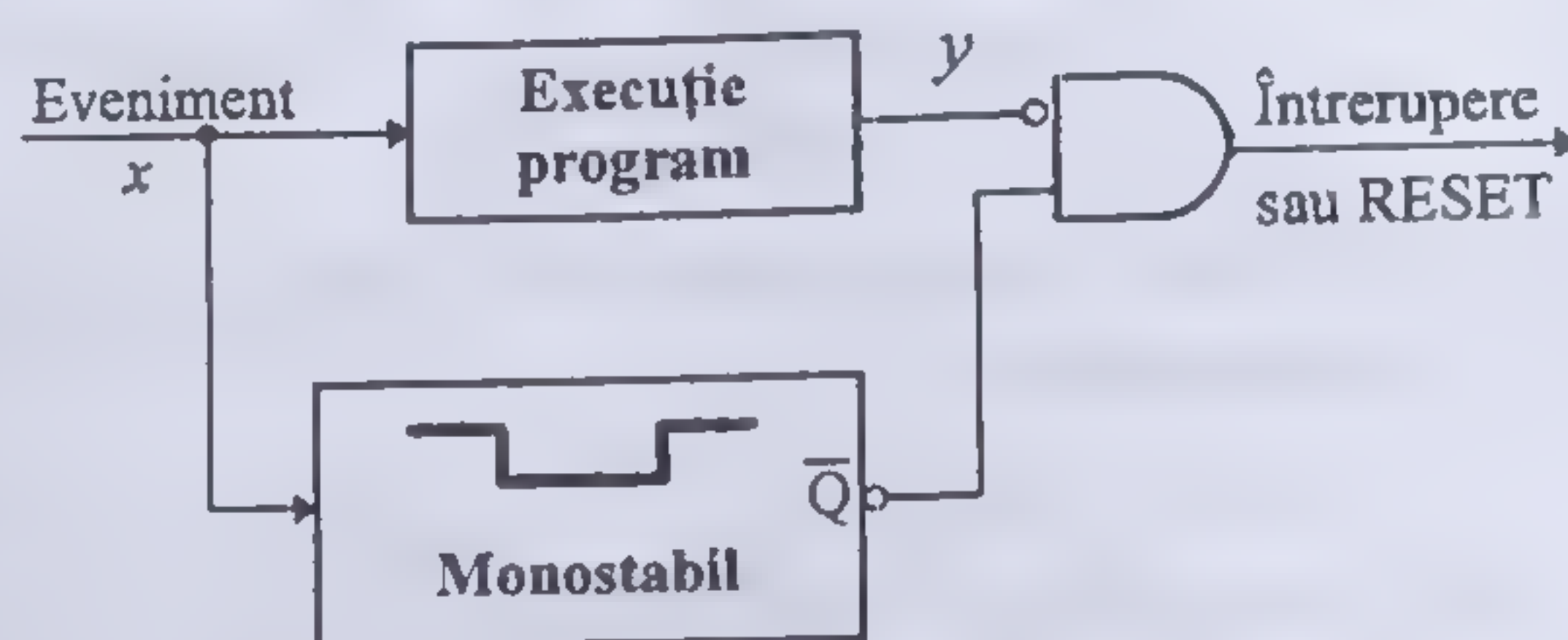


Fig.2.11. Schema de principiu a ceasului de gardă

stabilului în "0" logic, pe o durată de timp Δt . Astfel, ieșirea porții ȘI este forțată în "0" logic. Dacă durata execuției programului este mai mică decât Δt , atunci y devine "1" înainte ca ieșirea \bar{Q} a monostabilului să treacă în "1" logic, deci ieșirea porții ȘI rămâne în continuare inactivă ("0" logic). Dacă, din cauza unei funcționări defectuoase, execuția programului durează mai mult decât perioada de întârziere a monostabilului (durata maximă așteptată), trecerea ieșirii \bar{Q} în "1" logic determină generarea unei cereri de întrerupere sau activarea semnalului de RESET de la ieșirea porții ȘI. Supravegherea timpului de răspuns (TIME OUT) al unor părți vitale dintr-un sistem de conducere, cum ar fi memoria principală sau anumite dispozitive de I/E, poate fi realizată folosind dispozitive de tip ceas de gardă.

Cea de-a treia cerință este asociată direct evoluției în timp real a proceselor fizice, implicând cunoașterea valorii exacte a *timpului astronomic* (anul, luna, ziua, ora, min., sec.) în orice moment, în vederea înregistrării apariției unor evenimente sau a momentului la care s-au elaborat și transmis anumite comenzi, ca reacție la aceste evenimente (time stamping). Astfel, acest mecanism, denumit și *ceas de timp real* (Real Time Clock - RTC), are un rol esențial la înregistrarea evoluției în timp a variabilelor urmărite din proces și a comenzilor date către acesta și poate servi și drept referință pentru calcularea unor intervale de timp.

Toate aceste tehnici de introducere a timpului pot fi implementate fie cu dispozitive de tip monostabil sau astabil (spre ex. 74SN123, β E555 etc.) - care marchează intervale de timp fixate cu ajutorul unor grupuri RC reglabile, fie folosind numărătoare programabile comandate cu semnale de tact derivând din tactul microprocesorului - care asigură o flexibilitate și o precizie mult mai bune. Astfel, aceeași structură logică, de tip numărător programabil, poate fi folosită atât pentru măsurarea timpului cât și pentru a îndeplini funcții de tip numărător de evenimente.

În prezent se utilizează circuite programabile LSI/MSI de tip numărător/temporizator, care se conectează direct pe magistralele sistemelor cu microprocesoare și preiau rezolvarea taskurilor de timp ale acestora. Dintre cele mai cunoscute circuite de acest tip se pot menționa PIT8253/8254 (Programmable Interval Timer), utilizate în special în sistemele cu microprocesoare Intel și CTC-Z80 (Counter/Timer Circuit), pentru sistemele cu microprocesoare Z80.



Comunicația serială în sistemele cu microprocesoare

Comunicația serială a fost folosită inițial cu scopul de a cupla terminale inteligente sau periferice standard (videoterminale, imprimante etc.), care interacționează direct sau indirect cu operatorul uman, la un calculator central aflat la distanță. Datorită avansului tehnologic în domeniul microelectronicii, acest tip de comunicație s-a dovedit a fi adecvat și pentru interconectarea unităților centrale cu microprocesoare în cadrul sistemelor de conducere distribuite. În ambele situații se pot pune în evidență două categorii de activități legate de realizarea unui sistem de comunicație serială: cele care vizează proiectarea și implementarea fizică, utilizând dispozitive specializate - *componenta hardware*, respectiv cele care au în vedere stabilirea unor algoritmi eficienți de transfer al datelor și integrarea lor în programe de comunicație - *componenta software*. Importanța și interdependența celor două componente impun cu necesitate o prezentare preliminară a problematicii comunicațiilor seriale utilizate în sistemele cu microprocesoare.

3.1. Elemente și concepte de bază

După cum s-a arătat în §1.1.2.1 și §1.1.3.1, pentru a transfera date binare între componentele interne ale unui sistem cu microprocesor se utilizează, datorită eficienței sale, formatul paralel pe o magistrală de date, sub controlul unor semnale de comandă. Aceeași procedură s-a extins și în exteriorul sistemelor cu microprocesoare, prin intermediul unor circuite de interfață paralelă, cu păstrarea unor performanțe ridicate în privința ratei transferului de date (v. cap.2). Dar, pe măsură ce distanța dintre sisteme crește, transferul paralel devine din ce în ce mai costisitor, datorită numărului mare de legături pentru cablarea celor două magistrale, iar perturbațiile și întârzierile diferite în propagarea semnalelor electrice pe linii lungi alterează performanțele transferului de date.

În astfel de situații se preferă o conversie a datelor din formatul intern - *paralel* - într-un format extern de tip *serial*. Datele binare sunt transmise bit după bit, pe aceeași linie cu semnalele de comandă aferente, conform unui *protocol de comunicație*. Același protocol precizează și modul în care datele în format serial, recepționate la destinație, vor fi reconstituite și reconvertite în format paralel.

Toate operațiile asupra datelor, atât la transmisie cât și la recepție, sunt efectuate de circuite dedicate, de tip interfață serială. Deci, prin comunicația serială se asigură transferul de date între două sau mai multe interfețe seriale situate la distanță una față de alta, distanță care poate varia de la câțiva metri până la valori foarte mari (localități diferite).

3.1.1. Protocele de comunicație

O colecție de reguli care trebuie să fie respectate de componentele hardware și/sau software și care controlează transferul de date într-un sistem de comunicație este denumită *protocol de comunicație*. Aceste reguli sunt convenite pentru a asigura eficiența și buna desfășurare a transferului de date. Într-un cadru mai larg, al comunicației de date în rețele de calculatoare, transferul de date se realizează sub controlul unui set de protocele de comunicație standard, organizate într-o structură ierarhizată, multistrat. Fiecare protocol grupează împreună funcții înrudite prin natura operațiilor pe care le reglementează. Pentru fiecare strat există protocele

standard, ale căror specificații trebuie respectate, din motive de compatibilitate, de fiecare participant la transferul de date. Spre exemplificare, în fig.3.1 se prezintă cele 7 straturi ierarhice ale modelului de referință OSI (Open Systems Interconnect) al ISO (International Standards Organisation), care este standardul general acceptat în domeniul comunicațiilor de date.

Aplicație	Aplicație
Prezentare	Prezentare
Sesiune	Sesiune
Transport	Transport
Rețea	Rețea
Legătură de date	Legătură de date
Fizic	Fizic
Mediu fizic de comunicație	

Fig.3.1. Structura ierarhizată a modelului de referință OSI

Pentru realizarea comunicației seriale este suficientă de obicei utilizarea unei structuri ierarhizate simplificate a modelului de referință OSI, formată din numai trei straturi: cel fizic, cel al legăturii logice de date și cel de aplicație, evidențiate și în fig.3.1.

Protocolul fizic (sau electric) este alcătuit dintr-un set de funcții, de obicei implementate în hardware, ce permit conversia datelor în semnale care se propagă prin *mediul fizic de comunicație*. Protocolul fizic permite controlul accesului la mediul de comunicație (Media Access Control - MAC). Unitatea informațională cu care se lucrează la acest nivel este *blocul de date*. În cazul comunicației seriale, acesta poate fi format din unul (protocoale *asincrone*) sau mai multe caractere (protocoale *sincrone*). Protocolul legăturii fizice stabilește doar condițiile necesare pentru transferul serial al unui bloc de date prin mediul de comunicație, fără a garanta întotdeauna corectitudinea transferului. Acest aspect este avut în vedere de nivelul ierarhic superior, de *control al legăturii de date*, care este prevăzut cu mecanisme adecvate de detectare și, eventual, de corectare automată a erorilor ce pot apare la recepție.

Protocolul legăturii de date grupează funcții de control asupra logicii transferului de date (Data Link Control - DLC), implementate în structuri hardware specializate sau prin program, în vederea:

- stabilirii sau întreruperii legăturii logice de date între transmițător și receptor;
- ajustării fluxului de date transmis, conform posibilităților de prelucrare și de stocare a sistemului destinație (sincronizarea între partenerii de comunicație);
- asigurării integrității mesajelor (detectie erori, retransmisii, confirmări);
- identificării transmițătorului și a receptorului etc.

Comunicația la nivelul legăturii de date se face prin *mesaje*. Un mesaj poate fi format din unul sau mai multe blocuri de date, încadrate între caractere cu rol de delimitatori. În cazul în care mediul de comunicație este utilizat numai de două echipamente, datele transmise de unul vor fi, evident, recepționate de celălalt, dacă acesta din urmă este pregătit. În acest caz se stabilește o *legătură punct la punct*, în care canalul de comunicație, de obicei bidirecțional, este tot timpul la dispoziția celor două sisteme, iar sursa și destinația mesajelor sunt implicite. Toate acestea conduc la simplificarea sarcinilor care revin protocolului legăturii de date.

În general însă, la același mediu de comunicație se pot conecta mai multe sisteme, formând o *structură de tip rețea*, cu legături multiple (*legături multipunct*). La nivelul rețelei este necesară asigurarea *disciplinei de comunicație*. În acest caz, protocolul legăturii logice de date trebuie să asigure și arbitrarea accesului concurent la mediul de comunicație comun, pentru stabilirea legăturii logice între două sisteme. În acest sens, trebuie menționată și posibilitatea de a impune anumite restricții și reguli pentru efectuarea transferului de date, transformând schema de acces la linia de comunicație dintr-una stochastică într-una deterministă. Astfel, unul dintre sisteme va prelua rolul de sistem coordonator (master) și va planifica alocarea mediului de comunicație, pe

Partea I - Microprocesoare în conducerea automată a proceselor

rând, sistemelor subordonate (slaves). Acestea din urmă vor putea transmite mesaje către master și, prin intermediul lui, celorlalte sisteme subordonate.

În structurile de tip rețea mai evoluate apar și alte sarcini, legate de:

- comutarea și stabilirea traseului de urmat pentru blocurile de date prin structura interconectată a rețelei;

- asigurarea unor servicii de transport de nivel înalt;
 - garantarea integrității datelor la nivel de mesaj;
 - coordonarea interacțiunilor dintre programele de aplicație ce rulează pe sisteme diferite;
 - efectuarea unor transformări asupra datelor vehiculate (conversii de cod și de format între reprezentările specifice programelor de aplicație și cele mai eficiente pentru transfer).

Aceste sarcini sunt suficient de complexe pentru a face, fiecare, obiectul unui protocol separat; de aici și prezența unor straturi superioare suplimentare: transport, sesiune, prezentare (v.fig.3.1).

Protocolul de aplicație se află la ultimul nivel, cel mai ridicat. El furnizează utilizatorului o interfață software (Application Programming Interface - API), care oferă un set complet de servicii de comunicație de nivel înalt. Aceste funcții pot fi apelate direct din programele de aplicație, izolând astfel utilizatorul de detaliile complicate de prelucrare și transfer al datelor din straturile inferioare.

3.1.2. Modelul general al comunicației seriale punct la punct

În fig.3.2 se prezintă componentele modelului general al comunicației seriale dintre două sisteme cu microprocesor.

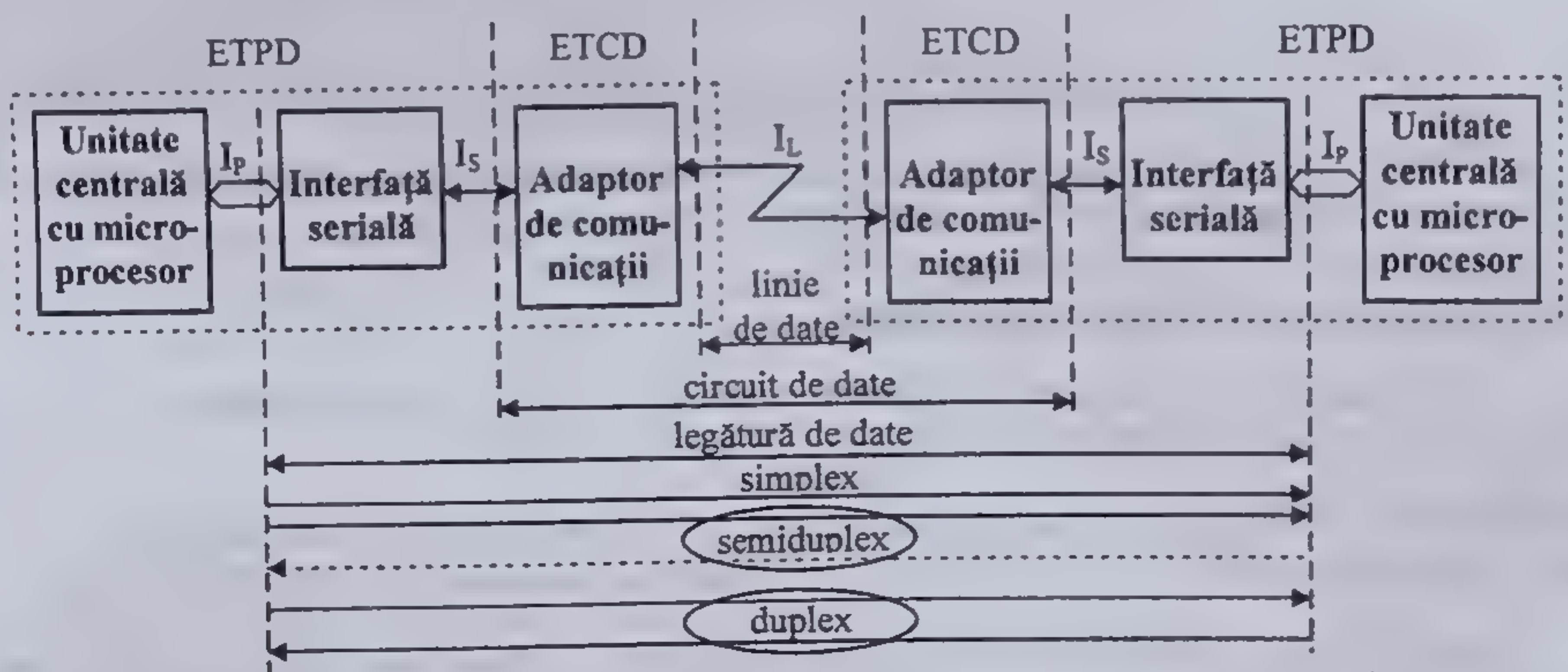


Fig. 3.2. Modelul general al comunicației seriale punct la punct

Echipamentele terminale de prelucrare a datelor (ETPD sau DTE - Data Terminal Equipment) - pot fi microcalculatoare, terminale inteligente, în general surse și/sau destinații de date, având în componența lor o unitate centrală cu microprocesor și o interfață de comunicație serială (sau controler de comunicație). Datele sunt transferate între unitatea centrală și interfața serială în format paralel (I_p), rolul interfeței seriale fiind, în principal, acela de a executa conversia bidirecțională de format al datelor între I_p și formatul serial (I_s). În plus, tot la nivelul ETPD se execută detecția și, eventual, corecția erorilor apărute, de către protocolul legăturii logice de date. La rândul ei, interfața serială implementează acele funcții ale protocolului fizic care sunt independente de mediul de transmisie (linia de comunicație).

Echipamentele terminale pentru comunicații de date (ETCD sau DCE - Data Communication Equipment). Deoarece nu orice canal de comunicație poate fi folosit pentru transmisie în banda de bază, sunt necesare dispozitive suplimentare, de modulare/demodulare (modemuri), care realizează o adaptare electrică a datelor la caracteristicile liniei de comunicație. În cazul în care mediul de transmisie este analogic (linii telefonice), aceste echipamente efectuează operații de modulare a șirului de biți pe un semnal purtător la transmisie, respectiv de demodulare a semnalului analogic și de refacere a formatului inițial al datelor la recepție. Ele implementează acele funcții ale protocolului fizic care sunt specifice mediului de comunicație, la care modemul se conectează printr-o interfață standardizată, I_L .

Linia de comunicație - reprezintă mediul fizic prin care se propagă datele seriale. Se pot utiliza mai multe tipuri de linii de comunicație:

- linii fizice dedicate - linii special instalate pentru comunicația serială sau rezervate în acest scop: linie telefonică închiriată, linie telex, fibră optică, transmisie prin unde radio;
- linii telefonice din rețeaua publică (linii comutate).

Există și situații în care transmisia are loc în banda de bază, interconectând direct echipamentele terminale de prelucrare a datelor (ETPD), prin linii fizice cu *cuplaj în curent continuu*, fără modulare/demodulare (null modem). Astfel se întâmplă, de obicei, la conectarea unor *periferice seriale* (terminale de tip consolă, imprimante seriale) la un sistem cu microprocesor, sau la realizarea comunicației seriale între două sisteme cu microprocesor, pe distanțe relativ mici - de ordinul zecilor de metri. În astfel de cazuri, circuitul de date degenerază în linia de date.

Legătura dintre interfața de comunicație serială și adaptorul de comunicație este standardizată din punct de vedere electric și mecanic, cel mai cunoscut și mai utilizat standard fiind EIA RS-232 (Electronic Industries Association), echivalent cu CCITT - V.24 (Comité Consultatif Internationale Téléphonique et Télégraphique). Alte standarde folosite: RS - 423A, RS - 422A și RS-485, care extind limita maximă de viteză a standardului RS-232.

Datorită importanței pe care o au în implementarea corectă a unui sistem de comunicație serială, standardele menționate mai sus vor fi prezentate pe larg în §3.2.

3.1.2.1. Parametrii comunicațiilor seriale

O caracteristică importantă a fluxului de date la nivelul I_S o reprezintă *viteza sau rata de comunicație*. Aceasta are două forme de reprezentare:

Debitul binar : D = numărul maxim de simboluri binare transmise într-o secundă [bps]

Viteza de comunicație: $V_c = 1 / (\text{intervalul de timp dintre două tranziții electrice})$ [Baud].

Deși, la prima vedere, ele par să însemne același lucru, există cazuri în care o tranziție electrică poate să corespundă la mai mult de un singur bit de date (cazul codificării informației), când cele două forme de reprezentare au valori diferite. Spre exemplu, dacă semnalul se modifică la fiecare 3,33ms, $V_c = 10^3 / 3,33 = 300$ Baud și nu este identică cu $D=300$ bps.

Pe de altă parte, se constată că un bloc de date este transmis în realitate cu o *viteză sau rată de comunicație efectivă* care este mai mică decât cea teoretică, exprimată anterior, din cauza faptului că pe aceeași linie se transmit nu numai biți de date, ci și biți reprezentând semnale de comandă și de sincronizare. De asemenea, pot să apară pauze în transmisie, timp în care linia e utilizată pentru confirmări, retransmisii și alte operații auxiliare specifice protocolului de comunicație utilizat, ceea ce reduce și mai mult viteza de comunicație serială.

La nivelul I_L informația este reprezentată de către modem prin stări electrice diferite, asociate stărilor logice de la intrare sau combinațiilor acestora, prin modulație în amplitudine, în frecvență, în fază sau prin modulație combinată. Este necesară utilizarea aceluiași tip de modem la

ambele capete ale liniei de transmisie. Fiecare stare electrică este menținută la ieșirea modemului un anumit interval de timp, numit *interval de modulație* (Δ). Astfel, la nivelul semnalelor emise/recepționate de modem se poate defini viteza de modulație (V_m), ca fiind numărul de schimbări pe secundă ale stării electrice la ieșirea modemului (inversul intervalului de modulație), care se măsoară în [Baud].

$$V_m = \frac{1}{\Delta} [\text{Baud}]$$

Relația dintre D și V_m este: $D = V_m \log_2 n$ [bps], unde n este numărul stărilor electrice distincte utilizate de modem.

Comunicațiile seriale pot fi clasificate după nivelul la care se face sincronizarea transferului de date în: comunicații *asincrone* (la nivel de caracter), respectiv *sincrone* (la nivel de bloc de caractere). Legăturile de date seriale pot fi, în funcție de sensul de comunicație, de tip *simplex* - flux de date unidirecțional, *semi-duplex* - flux de date în ambele sensuri, dar nu simultan, respectiv *duplex* - flux de date simultan bidirecțional, așa cum se observă și în fig.3.2.

Valorile vitezelor (ratelor) de comunicație sunt standardizate, pentru transmisiile asincrone uzuale fiind folosite valorile: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400 [Baud], iar debitele binare pot lua valori din același șir, precum și valori intermediare ca 3600, 7200, 14400, 28800 [bps].

Comunicația asincronă este utilizată în special în cazul dialogului om-mașină, prin intermediul unor dispozitive periferice standard, sau mașină-mașină atunci când este suficientă o viteză relativ redusă. De obicei, o astfel de comunicație necesită legături simple, de tip punct la punct. Comunicația sincronă acoperă domeniul transferului de date mașină-mașină, la viteze relativ ridicate și de cele mai multe ori se utilizează în legături seriale multipunct.

3.1.3. Modelul general al comunicației seriale multipunct

În fig.3.3 se prezintă componentele modelului general al comunicației seriale între mai mult de două sisteme cu microprocesor, în care fiecare dintre sistemele componente poate comunica cu oricare din celelalte sisteme, folosind o singură interfață de comunicație. Este cazul tipic al unei rețele de sisteme cu microprocesoare, interconectate printr-o magistrală serială (serial bus).

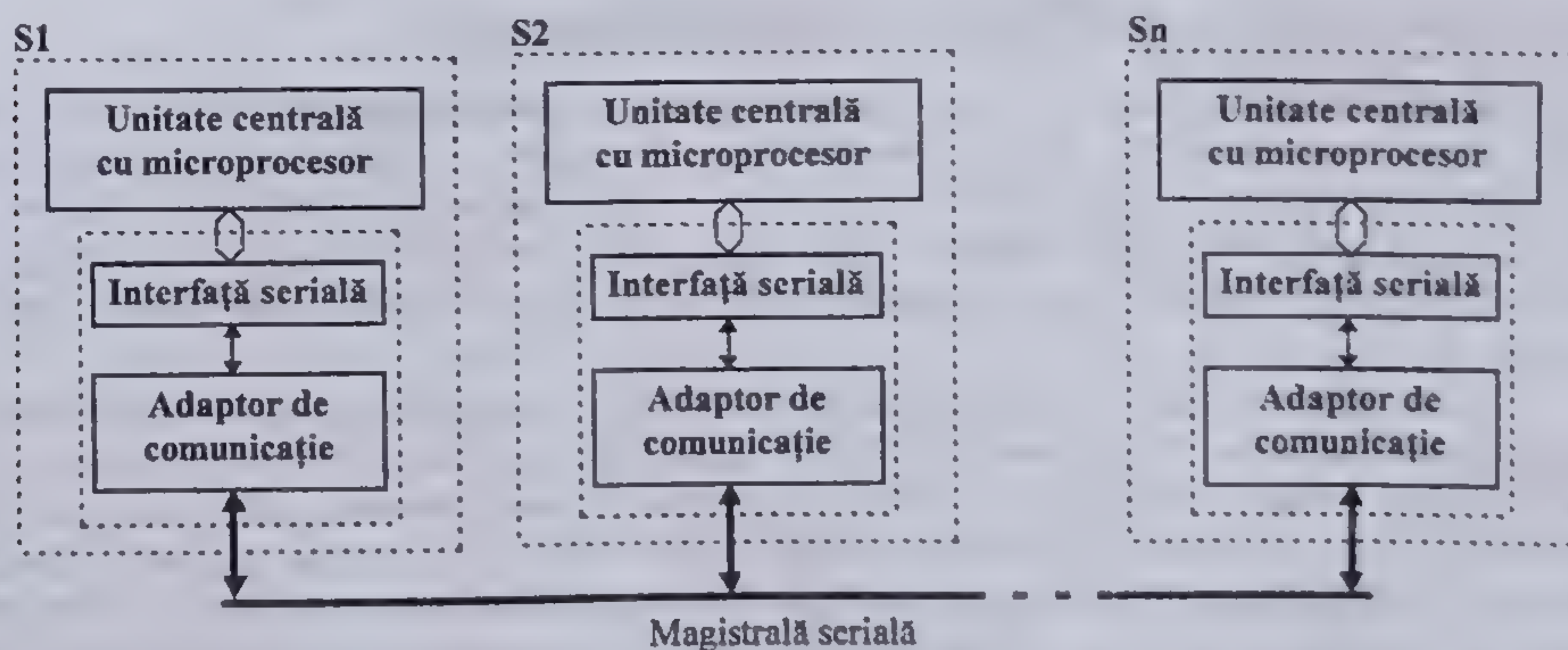


Fig. 3.3. Modelul general al comunicației seriale multipunct

Dacă pe toate sistemele interconectate se execută același protocol de comunicație la nivelul legăturii de date, fiecare sistem va funcționa independent de celălalt și deci încercările de a

transmite date pe magistrală vor fi asincrone unele față de altele. Astfel, pot să apară tentative de transmisie simultane de la două sau mai multe sisteme, ori poate fi întreruptă o transmisie în curs a unui sistem de către alte sisteme, care doresc și ele să transmită date. Aceste situații sunt desemnate cu termenul generic de *coliziuni*. Deoarece canalul de comunicație este unic, nu suportă decât situația în care, la un moment dat, un singur sistem transmite date pe linie și unul sau mai multe din celelalte sisteme recepționează aceste date. Coliziunile au un efect distructiv asupra datelor transmise, prin interferența semnalelor emise simultan pe aceeași linie fizică. Eficiența globală a unui astfel de protocol de comunicație, complet *stochastic*, este foarte scăzută și, de aceea, în practică se utilizează variante îmbunătățite ale acestei scheme.

Se ține cont de faptul că încercările de transmisie ale unui sistem, atunci când un alt sistem deține deja controlul liniei de comunicație, au o probabilitate de apariție mult mai mare decât încercările simultane de transmisie, a două sau mai multe sisteme, pe linia de comunicație liberă. De aceea, adaptoarele de comunicație sunt prevăzute cu mecanisme de *ascultare* (listening) a liniei, care permit începerea unei transmisii numai dacă aceasta este liberă (dacă nu există o transmisie în curs de desfășurare). De asemenea, pentru a abandona temporar transmisia în cazul în care apar încercări de transmisie simultane pe linia liberă, adaptoarele de comunicație sunt prevăzute cu mecanisme de *detectare a coliziunilor*. În cazul în care a fost detectată o coliziune a două sau mai multe sisteme, acestea încetează transmisia și vor încerca să o reia după scurgerea unor intervale de timp aleatoare, pentru a minimiza probabilitatea de a intra din nou în coliziune.

Această metodă de acces la mediul de comunicație este cunoscută sub numele CSMA/CD (Carrier Sense Multiple Access with Collision Detection) sau LBT (Listen Before Talk) și are numeroase variante de implementare. Și în acest caz comunicația păstrează un pronunțat caracter stochastic, ceea ce face să nu poată fi determinată o limită minimă, garantată, a intervalului de timp în care un sistem transmite un mesaj către un alt sistem din rețea. De asemenea, adaptorul de comunicație are o structură complexă și de aceea este implementat sub forma unui *adaptor de rețea* (transceiver), care înglobează interfața serială și poate conține chiar și un coprocesor specializat pentru funcții de comunicație.

Această variantă este acceptabilă în cazul în care pe magistrala comună se vehiculează un număr relativ mic de blocuri de date, de dimensiuni mari, iar aplicațiile care rulează pe sistemele interconectate nu lucrează în timp real.

3.1.3.1. Comunicația serială multipunct de tip master/slave

În aplicațiile de conducere în timp real a proceselor, sistemele comunică între ele printr-un număr relativ mare de mesaje, de dimensiuni reduse, care trebuie să ajungă la destinație într-un interval de timp minim garantat. În astfel de cazuri este indicat să se utilizeze o schemă de comunicație *deterministă*. Ea conduce la simplificarea adaptorului de comunicație, deoarece o parte din funcțiile lui vor fi preluate și rezolvate prin program, la nivelul protocolului legăturii de date.

Un astfel de model se poate obține, de exemplu, dacă delegăm unuia din sistemele din rețea rolul de *coordonator* sau *master* (SC în fig.3.4). El execută o variantă de protocol de comunicație care diferă de cele rulate pe celelalte sisteme, numite sisteme *subordonate* sau *slave* (SS_i , $i=1, 2, \dots, n$). Sistemul coordonator are rolul de *arbitru de trafic* pe magistrala comună. Pentru a-și îndeplini acest rol, el poate acorda sistemelor subordonate accesul pe magistrală, folosind fie metoda de *interogare*, fie metoda *accesului concurențial* sau *prin competiție*.

Prima metodă se referă la explorarea ciclică, de către master, a sistemelor slave, pentru verificarea existenței solicitărilor de transmisie. Sistemul master dispune de o *listă circulară* (inel logic) formată din toate sistemele slave. Pentru început, sistemul master transmite un cod de comandă de *interogare* către un sistem slave. Dacă sistemul slave interogată are date de transmis,

atunci răspunde transmițând datele către master. În caz contrar, răspunde cu un cod de înfirmare. Dacă datele primite de master trebuie transmise către un alt sistem, atunci sistemul master își îndeplinește rolul de intermediar și asigură livrarea datelor la destinație. Această procedură se repetă cu fiecare din sistemele slave, în ordinea fixată de lista circulară deținută de sistemul master, într-o buclă infinită.

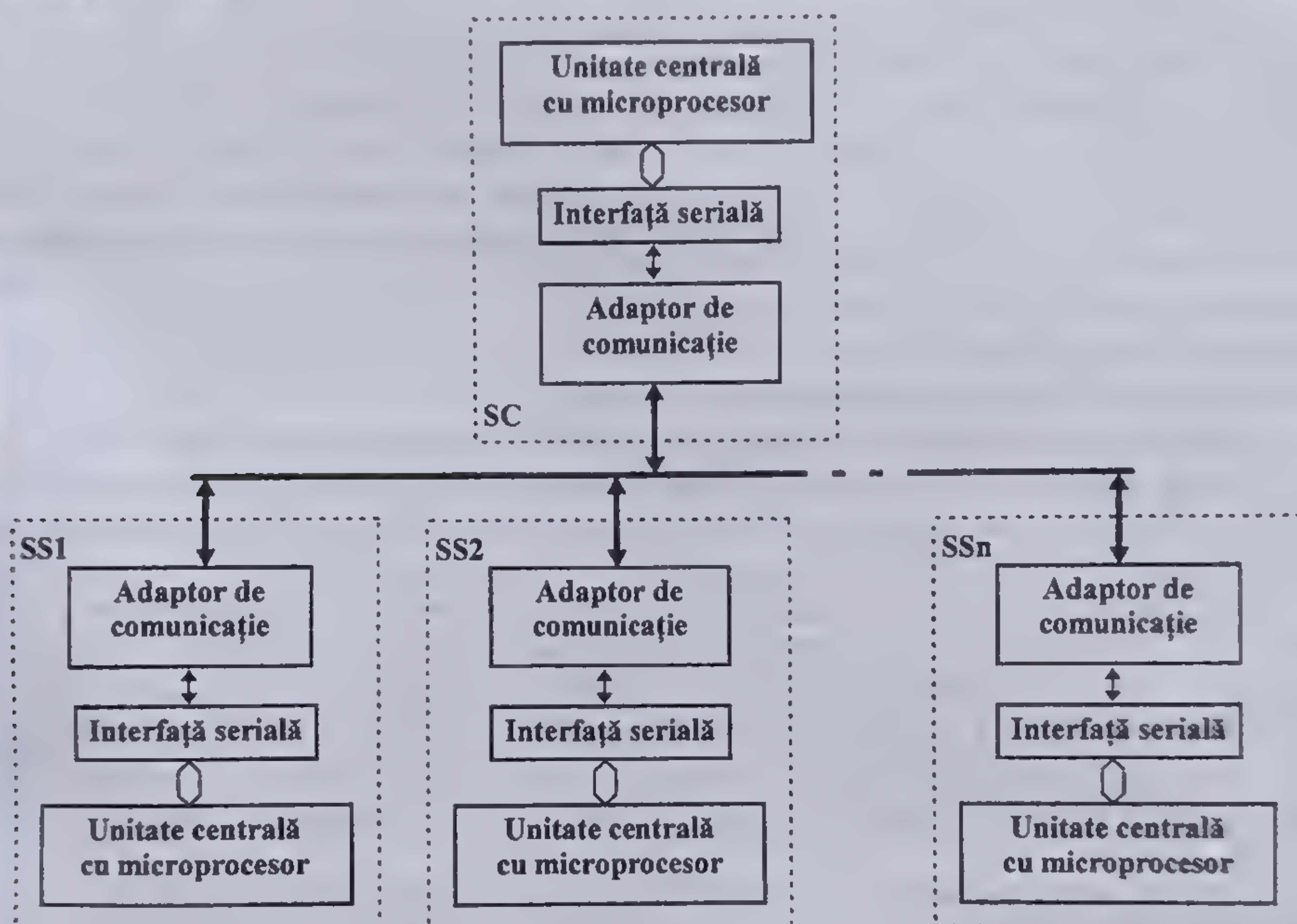


Fig.3.4. Modelul comunicației seriale multipunct de tip master/slave

Se observă că nu există un dialog direct între dispozitivele slave, ci numai prin intermediul dispozitivului master. Deși, din acest punct de vedere, prezintă dezavantajul unei dublări a timpului de transmisie a datelor, metoda are avantajul că garantează transmiterea unui mesaj între două sisteme într-un interval de timp limită cunoscut. Acesta depinde de numărul n de sisteme din rețea, de viteza de comunicație și de dimensiunea maximă admisă pentru un mesaj.

Pentru a limita degradarea performanțelor modelului odată cu creșterea numărului de sisteme slave, se poate generaliza structura din fig.3.4 sub o formă de arbore, pe mai multe niveluri, prin gruparea unui număr de sisteme slave în jurul unui *nod* de tip master. La rândul lor, un număr de noduri de tip master vor putea fi grupate și vor conta ca noduri slave pentru un *supernod* de tip master la care vor fi conectate ș.a.m.d., până la nodul *rădăcină*.

A doua variantă, cea de acces concurențial, presupune alocarea de priorități (fixe sau variabile) celor n sisteme subordonate și acordarea accesului la magistrală celui mai prioritar dintre sistemele care l-au solicitat. În acest caz, ordinea de interogare este ordinea descrescătoare a priorităților la un moment dat. Dacă un sistem slave interogată are date de transmis, acesta le trimite către master care, eventual, le transmite mai departe sistemului slave destinație. Apoi, sistemul master reia interogarea, de la cel mai prioritar sistem: același de fiecare dată, dacă se lucrează cu priorități fixe, sau cel care a rezultat în urma actualizării periodice a priorităților, în conformitate cu un anumit criteriu. Dacă schema de modificare a priorităților este nedeterministă, sistemele slave sunt interogate aleator, ceea ce face să nu mai putem vorbi de acel interval limită minim

garantat de livrare a mesajelor de la sursă la destinație. Se păstrează astfel caracterul determinist al accesului la mediul de comunicație, dar numai din punctul de vedere al protocolului de la nivelul fizic, nu și din cel al protocolului la nivelul legăturii de date. Și în acest caz se menține dezavantajul unei transmisii duble a aceluiași mesaj pe traseul: slave sursă - master - slave destinație.

Indiferent de metoda de interogare utilizată, selectarea unui sistem slave din cele n se face cu ajutorul unui cod de identificare (de obicei, un octet), numit *adresă*. La interogare, toate sistemele slave primesc simultan un astfel de cod, dar numai unul din ele va răspunde: cel a cărui adresă este egală cu cea recepționată. Pentru a se preveni blocarea mecanismului de interogare, în cazul în care, din diverse motive, un sistem slave interogată nu răspunde într-un interval de timp limită, se consideră că acel sistem este defect și se iau măsurile care se impun (excludere temporară din lista circulară), după care se continuă lucrul cu restul sistemelor aflate în stare de funcționare. Deși aceasta contribuie la creșterea fiabilității sistemului pe ansamblu, el rămâne în continuare deosebit de vulnerabil la o cădere a sistemului master, când subsistemele, deși în continuare interconectate fizic, rămân izolate din punct de vedere logic.

Pentru a elimina un astfel de blocaj, ar trebui ca la detectarea scurgerii unui interval limită de timp fără primirea unei comenzi de interogare, unul din sistemele slave să fie capabil să preia rolul de coordonator, în locul celui defect. *Regenerarea* coordonatorului se poate face, de exemplu, prin menținerea, la nivelul fiecărui sistem slave, a unei constante de timp care să indice timpul maxim admis între interogări. La atingerea acestei limite sistemul slave respectiv trebuie să preia inițiativa de a se transforma în master. Aceste constante pot fi, de exemplu, dependente de prioritatea fiecărui sistem slave dintr-un sistem de acces concurențial sau, în general, pot fi orice set de n valori care diferă între ele printr-o valoare constantă. Această valoare constantă se alege ca fiind cel puțin egală cu intervalul de timp necesar unui sistem slave pentru a se transforma în sistem master, pentru ca nu cumva două sisteme slave să ia decizia de a se transforma în master în același timp.

Rezultă astfel necesitatea ca toate sistemele interconectate să fie echivalente din punctul de vedere al comunicației, deci să execute același protocol, cu singura deosebire că unul dintre sisteme (și numai unul) îndeplinește *temporar* rolul de master. De aici apare imediat ideea că această *funcție de master* ar putea circula de la un sistem la altul în mod normal, nu numai atunci când se defectează coordonatorul curent. Principalul avantaj al unui astfel de sistem este acela că elimină duplicarea transferului de date întâlnită la sistemele cu master fix și conduce la creșterea eficienței comunicației.

Astfel, sistemul care deține la un moment dat funcția de master, pe care în cele ce urmează o vom numi *jeton* (token), deține controlul magistralei comune. Aceasta înseamnă că, dacă are nevoie, poate să transmită date către oricare alt sistem conectat pe magistrală. În momentul în care nu mai are date de transmis, el transmite jetonul unui alt sistem, pe care îl alege după un criteriu oarecare, dintre sistemele interconectate. De obicei, sistemele sunt considerate ca fiind echivalente din punctul de vedere al comunicației și atunci jetonul circulă într-o buclă închisă (sau un inel logic) de la un sistem la altul, trecând pe la fiecare sistem de un număr egal de ori, dar se pot imagina și alte scheme, care să trateze în regim preferențial anumite sisteme prin alocarea unor priorități fixe sau variabile în timp. Deci, fiecare sistem primește mai întâi jetonul pe linie (de exemplu adresa sa în cadrul rețelei) de la sistemul precedent, transmite eventual date către un alt sistem, apoi transmite pe linie jetonul către sistemul următor (adresa următorului sistem din inelul logic).

Și în acest caz trebuie să se rezolve o serie de probleme care să asigure inițializarea și menținerea funcționării corecte a modelului de comunicație descris, cum ar fi inițializarea jetonului la pornirea a cel puțin două sisteme interconectate, introducerea sau scoaterea unui sistem din

buclă, regenerarea jetonului în urma opririi sau a defectării sistemului chiar în momentul în care acesta deținea jetonul etc.

Toate aceste probleme fac obiectul unor protocoale de comunicație standardizate, de tip Token-Bus. Datorită avantajelor pe care le prezintă (unele din ele au fost expuse și aici), aceste tipuri de rețele sunt cele mai utilizate în sistemele distribuite de conducere a proceselor.

Există și variante hibride de interconectare, care îmbină conexiunile punct la punct, de la nivel fizic, cu conexiuni de tip master/slave - la nivel logic, dintre care putem aminti structurile de tip *stea* și *inel*, mai puțin utilizate în rețelele de sisteme cu microprocesoare destinate mediului industrial.

3.1.4. Codificarea informației în comunicațiile seriale

Protocoalele legăturii de date utilizează simboluri binare (cuvinte de cod) atât pentru date cât și pentru comenzile necesare transferului serial al datelor. Protocoalele legăturii fizice de date nu fac nici o diferențiere după conținutul semantic al acestor simboluri. În funcție de codul utilizat, un cuvânt de cod se reprezintă prin 5 până la 8 biți. Exemple de astfel de coduri: codul Baudot (CCITT nr. 2) - cod telegrafic cu 5 biți/caracter utilizat de sistemul Telex, codul ASCII (CCITT nr. 5) cu 7 biți/cuvânt, codul EBCDIC cu 8 biți/cuvânt etc.

3.1.4.1. Detecția și corecția erorilor

Oricare din sistemele de comunicație descrise mai sus se bazează, în ultimă instanță, pe transmiterea de mesaje de la un sistem sursă la un sistem destinație. Imperfecțiuni ale canalului de comunicație, perturbații electromagnetice sau alte defecte hardware pot produce distorsiuni ale conținutului informațional al mesajului, deci pot introduce erori în mesajul recepționat. Utilizarea de către un sistem a unor date recepționate incorect poate conduce la erori de calcul și la comenzi neadecvate sau chiar periculoase pentru procesul condus.

De aceea se impune cu necesitate, pe de o parte creșterea gradului de imunitate la perturbații al mediului de comunicație, iar pe de altă parte *detectarea* și, eventual, *corectarea erorilor* din mesajul recepționat.

Detecția și corecția erorilor este o problemă esențială în comunicațiile de date și se poate face fie la nivelul protocolului legăturii de date, fie chiar la nivelul protocolului legăturii fizice. Metoda cea mai des folosită este *redundanța informațională*, adică adăugarea la transmisie a unor informații suplimentare de control, obținute prin diverse operații asupra elementului protejat (cuvânt, bloc, mesaj). Acest lucru permite efectuarea la recepție a unor verificări asupra corectitudinii datelor primite și, eventual, chiar o corecție automată, dacă metoda de codificare folosită o permite. Metodele cele mai uzuale de detectare a erorilor, prevăzute în protocoalele nivelului de control al legăturii de date, sunt următoarele:

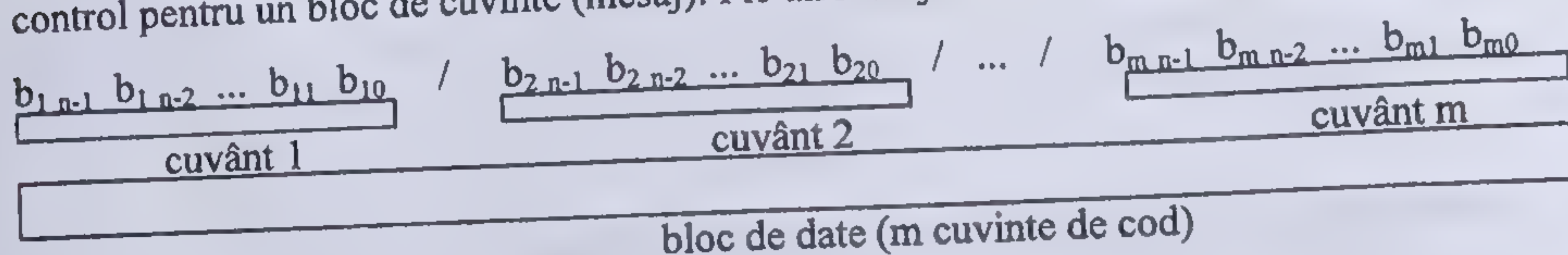
Metoda VRC (Vertical Redundancy Check) - generare/verificare paritate la nivel de cuvânt de cod, constă în adăugarea unui bit suplimentar, b_n , la cei n biți de date, $b_{n-1} b_{n-2} \dots b_1 b_0$, ai cuvântului. Acest bit se calculează din relația $\sum_{i=0}^n b_i = 0$, în cazul VRC cu paritate pară, sau din

relația $\sum_{i=0}^n b_i = 1$, în cazul VRC cu paritate impară. La transmisie, un bit de paritate calculat

conform uneia din relațiile de mai sus este inserat după biții de date ai cuvântului. La recepție se recalculează bitul de paritate al cuvântului primit, cu aceeași relație, și se compară apoi cu bitul de paritate recepționat. Dacă cei doi biți nu sunt identici, cuvântul recepționat este cu certitudine diferit de cel transmis, deci este eronat.

Metoda detectează erorile simple, de un bit într-un cuvânt (cele mai probabile) și, în general, orice modificare a unui număr impar de biți ai caracterului transmis. Nu pot fi detectate însă erorile duble și nici celelalte erori în care sunt modificați un număr par de biți ai aceluiași cuvânt de cod.

Metoda LRC (Longitudinal Redundancy Check) - generare/verificare a informațiilor de control pentru un bloc de cuvinte (mesaj). Fie un mesaj format dintr-o secvență de m cuvinte:



Se calculează un cuvânt $L = l_{n-1} l_{n-2} \dots l_1 l_0$, astfel încât $\sum_{i=1}^m b_{ij} + l_j = 0, \forall j = 0, 1, \dots, n-1$, în

cazul LRC pară, sau $\sum_{i=1}^m b_{ij} + l_j = 1, \forall j = 0, 1, \dots, n-1$, în cazul LRC impară. Transmițătorul

calculează pentru un bloc de date un cuvânt L , conform uneia din relațiile de mai sus, pe care îl transmite pe linie după caracterele blocului. Receptorul calculează un cuvânt L' pentru blocul de date recepționat, cu aceeași relație utilizată și la transmisie, pe care îl compară apoi cu cuvântul L recepționat. Dacă $L' = L$, se consideră că blocul de date recepționat este identic cu cel transmis (transfer fără erori), altfel, se consideră că blocul de date recepționat este eronat. Metoda detectează erorile simple de un bit, apărute pe același rang într-un bloc de date și, în general, orice modificare a unui număr impar de biți, pe același rang, ai blocului transmis. Nu pot fi detectate însă erorile duble apărute pe același rang și respectiv nici celelalte erori în care sunt modificați un număr par de biți ai aceluiași bloc de date.

Metoda CRC (Cyclic Redundancy Check) - generare/verificare rest CRC pentru un bloc de cuvinte de cod (mesaj). Blocul de date este tratat ca un șir de biți, formând un singur cuvânt de date. Lui îi corespunde un polinom de cod în care rangurile descresc, pe cuvinte, de la începutul blocului către sfârșit, iar în cadrul fiecărui cuvânt de la cel mai semnificativ la cel mai puțin semnificativ bit. Polinomul de cod rezultat se împarte la un polinom de cod standard, numit polinom generator (G), cu proprietăți dovedite privind detecția și corecția erorilor. Restul împărțirii (R), numit *sindromul* sau *codul de redundanță ciclică* al polinomului de cod, este tot un polinom, de grad mai mic decât polinomul generator, ai cărui coeficienți formează codul de verificare al blocului de date inițial; biții acestui cod se transmit pe linie, imediat după ultimul bit al blocului de date. La recepție, blocul de date recepționat se împarte la același polinom generator. Dacă restul obținut este diferit de zero, aceasta indică prezența unei erori în mesajul recepționat.

Două din polinoamele generatoare cel mai des utilizate în comunicația serială sunt CRC-16 ($G = X^{16} + X^{15} + X^2 + 1$) și CCITT ($G = X^{16} + X^{12} + X^5 + 1$). Generarea și controlul cuvintelor de cod CRC se poate realiza *prin program* sau prin utilizarea unor structuri hardware, sub forma unor *registre de deplasare cu reacție*.

Spre exemplu, să considerăm cazul polinomului generator specific codului CRC-16, la transmiterea/recepția unui bloc de octeți (fig.3.5). Împărțirile se realizează prin deplasarea de la A către B a polinomului deîmpărțit, iar adunarea prin circuite de tip SAU EXCLUSIV. Numărul de ranguri al registrului de deplasare este egal cu gradul polinomului generator (16), iar numărul de porți SAU EXCLUSIV este mai mic cu 1 decât numărul de coeficienți "1" ai polinomului generator ($4-1=3$).

Inițial, registrul este inițializat fie cu biți "0", fie cu biți "1". La transmisie, șirul de biți de transmis este adus, bit cu bit, la intrarea A. Celulele registrului conțin astfel valoarea momentană a

restului CRC, la fiecare impuls de sincronizare. După ce și ultimul bit al șirului de date a trecut prin A, registrul conține restul CRC asociat întregului mesaj. El se transmite pe linie, imediat după ultimul bit al mesajului, prin ieșirea B. Aceasta este deschisă numai pe durata transmiterii restului

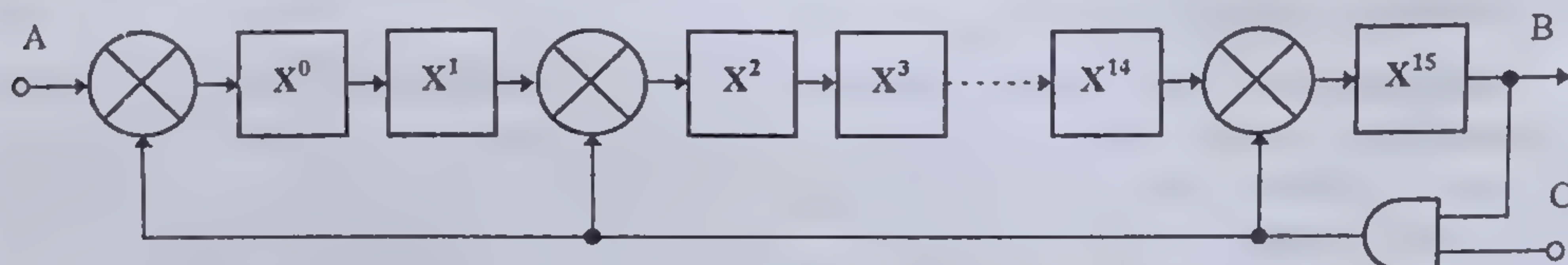


Fig.3.5. Implementarea hardware a generării restului CRC pentru polinomul $G = X^{16} + X^{15} + X^2 + 1$ (CRC-16)

CRC; transmisia propriu-zisă a mesajului se face pe o altă linie, care nu apare în fig.3.5. Schema funcționează ca un registru simplu de deplasare, reacția fiind inhibată prin blocarea porții ȘI cu $C=0$, pentru a nu altera restul CRC.

La recepție se utilizează un registru de deplasare cu aceeași structură cu a celui de la transmisie (specifică polinomului generator), inițializat la fel - cu biți "0" sau "1". Biții recepționați intră prin A și, după ce și ultimul bit din blocul de date a trecut prin A, urmează biții restului CRC. După ce și ultimul bit al restului CRC a trecut prin A, registrul trebuie să conțină același cod ca și după inițializare (numai biți "0" sau numai biți "1") pentru ca mesajul să fie considerat fără erori. Dacă se obține un alt cod, atunci se consideră că au apărut erori în timpul transferului.

Codurile de redundanță ciclică sunt utilizate pe scară largă în transferurile de date atunci când mediul de transmisie sau de stocare a informațiilor este susceptibil de a introduce erori. În contextul mai general, al codurilor liniare, care stau la baza primelor două metode (VRC și LRC), utilizarea codurilor ciclice conduce la simplificări importante ale operațiilor de detectare și corectare a datelor eronate.

Prin utilizarea codurilor detectoare de erori, mesajul este codificat la transmisie în așa fel încât simpla decodificare a lui la recepție să se poată constitui într-un test de validare. Dacă sunt detectate, erorile pot fi corectate *în mod automat* (coduri corectoare) sau prin retransmisia blocului eronat, până când nu se mai detectează nici o eroare.

Corecția prin retransmisie poate fi de trei tipuri:

- cu oprire și așteptare sau ARQ - ACK (Automatic Request for Repeat - Acknowledge), în care transmițătorul așteaptă confirmarea corectitudinii datelor după fiecare bloc transmis. În cazul unui răspuns afirmativ (ACK), transmite următorul bloc; în cazul unui răspuns negativ (NACK), se retransmite ultimul bloc.

- cu retransmisie continuă (ARQ - NACK) - se transmit blocurile de date succesiv, fără pauză între ele; în cazul sosirii unei înfirmări (NACK), se reia transmisia de la blocul recepționat eronat.

- cu retransmisie selectivă - asemănătoare cu (ARQ - NACK), cu deosebirea că se retransmite doar blocul recepționat cu erori.

3.2. Adaptoare de comunicație serială

Comunicația serială la distanțe mari nu se poate efectua cu semnale de nivel TTL, așa cum sunt cele generate de circuitele de interfață descrise în §2.2, datorită atenuărilor și perturbațiilor care apar pe linii de comunicație lungi. În plus, transferul de date dintre interfața de comunicație (ETPD) și adaptoarele de comunicație (ETCD) necesită și alte semnale de dialog, care asigură controlul fluxului de date dintre cele două dispozitive. Ele se regăsesc în fig.2.4 (cap.2) la nivelul blocului de comandă modem. Aceasta a condus la introducerea unor standarde de interconectare

pentru comunicații seriale, care precizează atât caracteristicile electrice ale semnalelor cât și modul de organizare și caracteristicile mecanice ale conectorilor și conexiunilor.

3.2.1. Standardul RS-232

Introduse în 1960, de către organizația *Electronic Industries Association* (EIA), specificațiile RS-232 erau la acea dată prima încercare de standardizare în domeniu. Ele au suferit în timp unele modificări, ajungându-se în situația ca varianta RS-232C să devină un standard *de facto* și să fie adoptată ca atare de instituțiile cu atribuții de standardizare din acest domeniu. Astfel, ISO l-a adoptat ca standard de interfațare cu ETCD de tip modem, iar CCITT a elaborat avizul V.24, care preia integral specificațiile EIA RS-232C.

Caracteristicile electrice ale acestui standard sunt prezentate în fig.3.6, unde se observă că domeniul $+3 \div +12V$ corespunde nivelului logic "0" (Low - L), iar intervalul $-12 \div -3V$ nivelului logic "1" (High - H). Banda $-3 \div +3V$ este zona logică interzisă. Impedanța de ieșire trebuie să fie de $0,1 \div 1k\Omega$, iar nivelul tensiunilor să nu depășească $\pm 12,5V$ (fără sarcină). O variantă a acestei interfețe este aceea în care se utilizează o buclă de curent, când specificația electrică a semnalelor se definește în curent: 20 mA sau 60mA pentru nivel H și curent nul pentru nivel L.

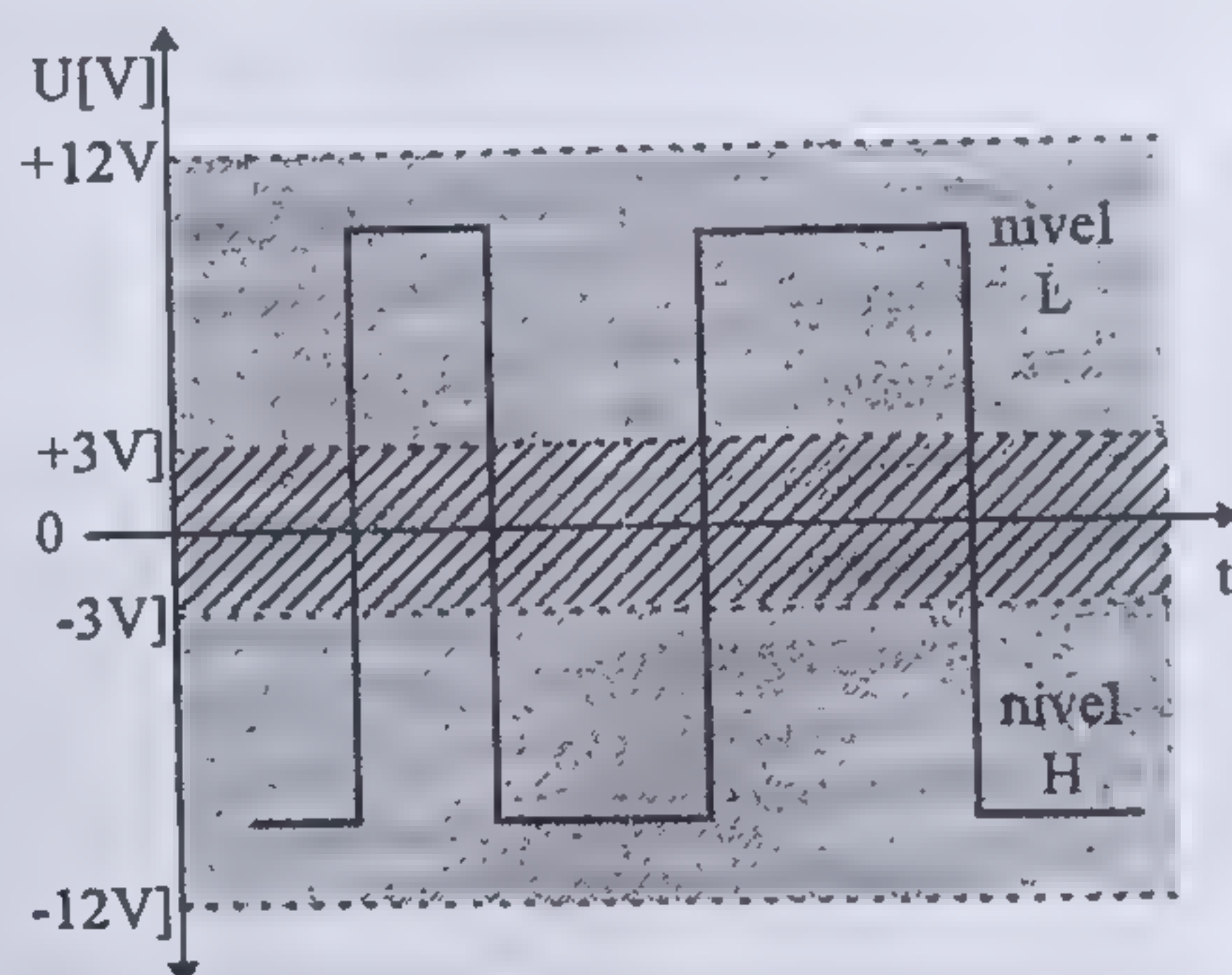


Fig.3.6. Caracteristicile electrice ale standardului RS-232

Standardul RS-232 definește și caracteristicile mecanice ale comunicației seriale: numărul de semnale și numele lor, tipul conectorilor și configurația semnalelor la pinii acestora. Astfel, sunt descrise funcțiile a 25 de semnale și linii de dialog pentru controlul transferului de date între interfața serială și modem. De asemenea, standardul prevede conectori de tip "tată" pentru DTE, respectiv de tip "mamă" pentru DCE. Nu este impus un anumit tip de conector, dar cel mai des utilizat este cel cu 25 de pini pe două rânduri, de tip DB-25P. Pentru sistemele la care multe din cele 25 de linii nu sunt necesare, se poate folosi un conector cu 9 pini, de exemplu de tip DE-9P. La realizarea legăturilor fizice DTE-DCE sau DTE-DTE trebuie să avem în vedere asignarea pinilor acestor conectori.

Pentru a înțelege mai bine modul în care se realizează transferul de date între interfața serială și modem, în fig.3.7 se prezintă, alături de schema bloc simplificată a unui modem, principalele semnale de dialog cu interfața serială (hardware handshake) precum și pinii la care se conectează, pentru ambele tipuri de conectori: cu 25, respectiv cu 9 pini.

La punerea sub tensiune a unității centrale și efectuarea tuturor inițializărilor necesare, interfața serială, sub controlul programului de comunicație, activează linia *Terminal de date pregătit* (\overline{DTR} - Data Terminal Ready) pentru a indica modemului că interfața serială este pregătită pentru comunicație. În mod similar, atunci când modemul este pregătit pentru comunicație, activează linia *Modem pregătit* (\overline{DSR} - Data Set Ready).

Sub control manual sau automat, modemul apelant încearcă să ia legătura cu echipamentul de la celălalt capăt al liniei. Dacă semnalul \overline{DTR} al modemului apelat este activ, la detectarea semnalelor de apel pe linie acesta activează semnalul *Indicator de apel* (RI - Ring Indicator), care va semnaliza sistemului apelat încercarea de stabilire a legăturii. Dacă acesta acceptă legătura, modemul apelat va returna un ton specific de confirmare.

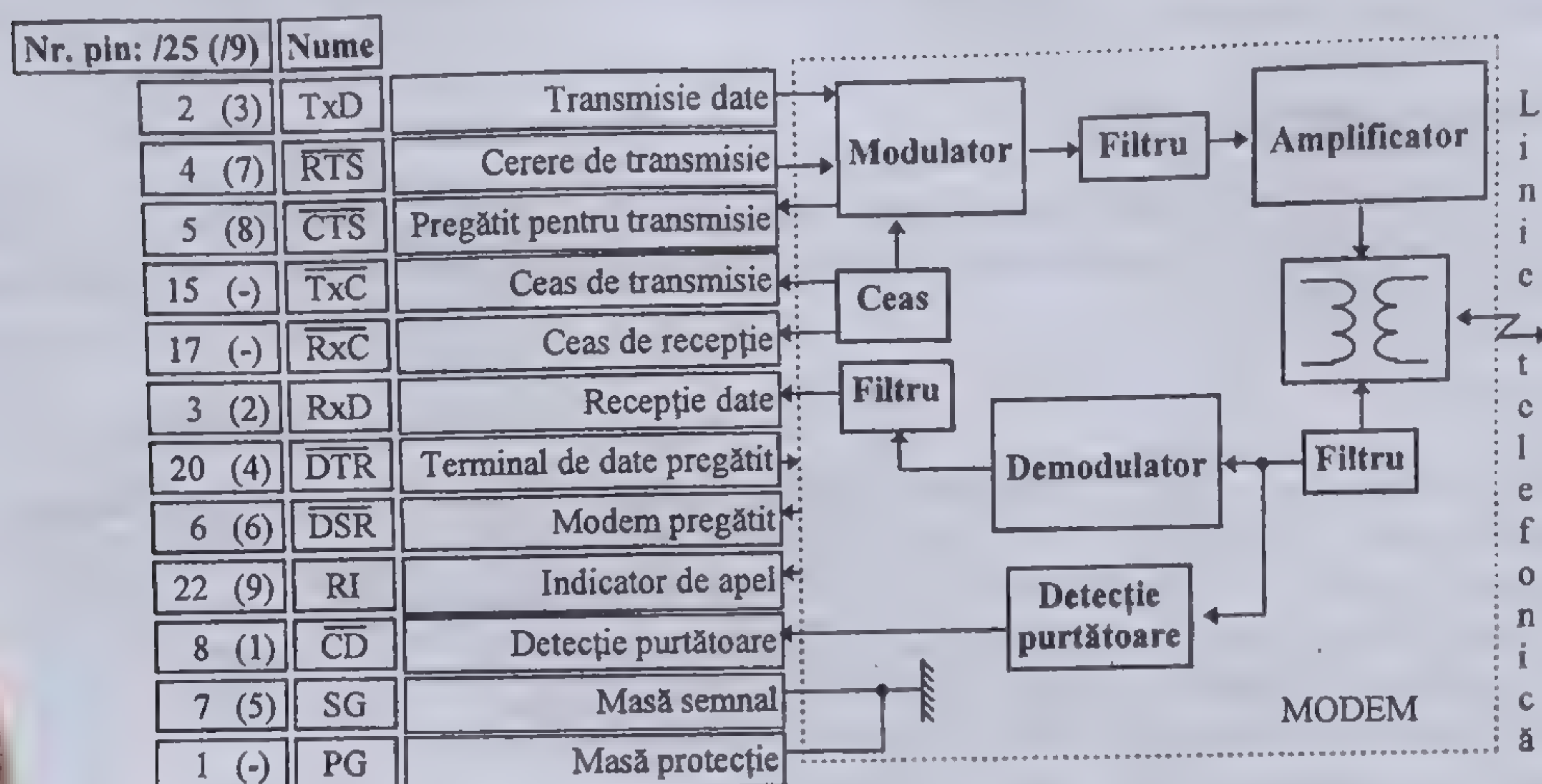


Fig.3.7. Structura generală a unui modem și semnalele standard RS-232C

Atunci când un sistem are date de transmis, el prezintă, prin intermediul interfeței seriale, o *cerere de transmisie* ($\overline{\text{RTS}}$ - Request To Send) către modemul asociat. Acesta transmite pe linie un semnal, pentru a informa sistemul destinație despre realizarea contactului, pregătindu-și circuitele proprii pentru transmisie, după care activează linia de confirmare *pregătit pentru transmisie* ($\overline{\text{CTS}}$ - Clear to Send), către interfața serială. În momentul primirii acestui semnal de confirmare ($\overline{\text{CTS}}=0$), transmițătorul interfeței seriale poate începe trimiterea datelor către modem, pe linia TxD, acestea fiind primite de către blocul modulator. După filtrare, semnalul este amplificat și trimis pe linia telefonică printr-un bloc separator cu transformator.

La recepție, blocul de dețecție a purtătoarei modemului apelat sesizează prezența semnalelor de stabilire a contactului și activează linia *prezență purtătoare* ($\overline{\text{CD}}$ - Carrier Detect), care pregătește interfața serială în vederea preluării datelor recepționate de modem. În continuare datele sunt preluate de pe linie prin intermediul aceluiași bloc separator cu transformator, sunt filtrate, demodate și, după o nouă filtrare, sunt trimise interfeței seriale pe linia RxD.

Transferul continuă până când sistemul apelant dezactivează linia $\overline{\text{RTS}}$, la care modemul asociat reacționează prin dezactivarea liniei $\overline{\text{CTS}}$ și prin eliminarea purtătoarei de pe linie. Modemul sistemului destinație detectează lipsa purtătoarei, își dezactivează la rândul lui linia $\overline{\text{CD}}$ și informează interfața serială despre încetarea transferului de date.

În fig.3.8 este prezentată schema de comunicație prin linie telefonică între două sisteme, DTE1 și DTE2, folosind modemuri (DCE1 și DCE2).

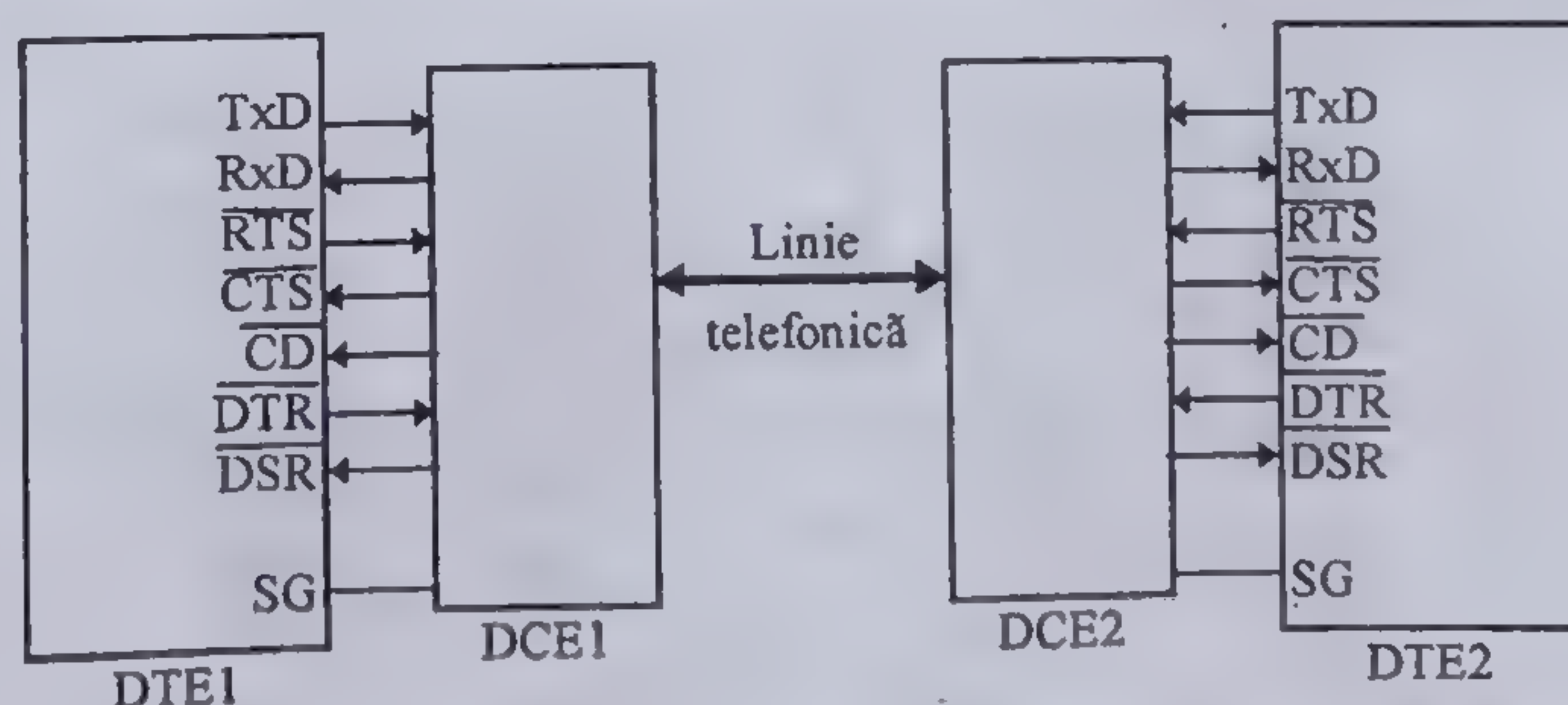


Fig.3.8. Comunicație serială pe linie telefonică folosind modemuri

Modul de control al transferului de date dintre ETPD și ETCD prin semnale de dialog se mai numește și *protocol RTS/CTS*.

3.2.1.1. Interfața TTL ↔ RS-232

Circuitele de interfață serială furnizează și pot primi semnale compatibile TTL; ele nu sunt direct compatibile cu caracteristicile electrice definite de standardul RS-232. Pentru adaptare se utilizează circuite de tip *emițător de linie MC1488* (TTL → RS-232), respectiv *receptor de linie MC1489* (RS-232 → TTL) sau echivalente ale acestora (SN74188, SN74189). Ele conțin câte 4 adaptoare pe chip, așa cum se arată în fig.3.9. Ieșirile driverelor RS-232 se conectează la masă, prin condensatoare, pentru a se obține timpii de comutație standard de $30\text{V}/\mu\text{s}$. Această soluție elimină și interinfluența (diafonia) între fire adiacente. La rândul lor, receptoarele au un pin care trebuie conectat la masă printr-un condensator de 1nF . Se poate observa că adaptoarele realizează și o inversare a nivelului logic, ceea ce explică alocarea nivelurilor logice din fig.3.6.

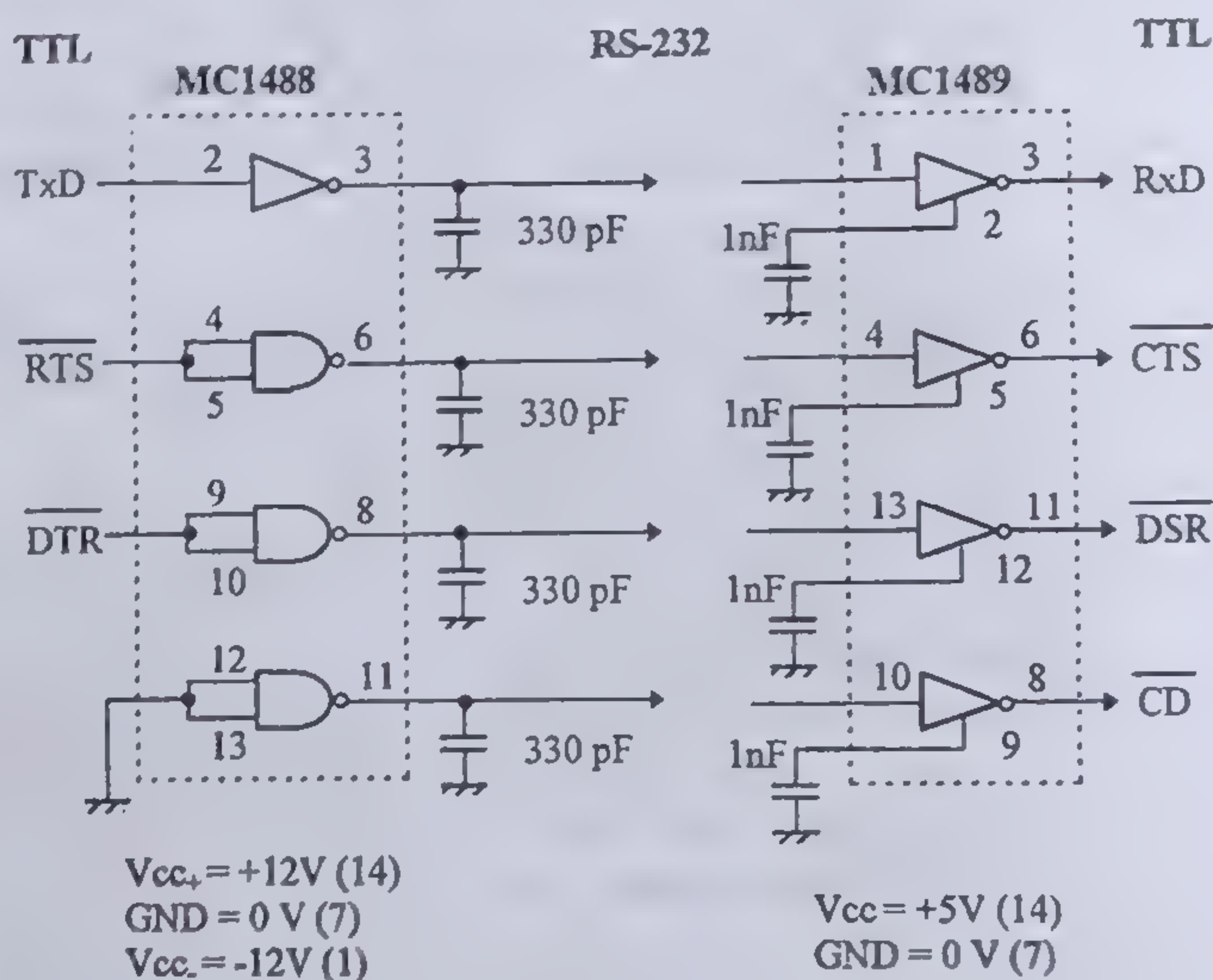


Fig.3.9. Conversia TTL ↔ RS 232 folosind circuite MC1488 și MC1489

3.2.1.2. Interconectarea echipamentelor compatibile RS-232

În fig.3.10a se prezintă modul de interconectare al unui echipament de tip DTE cu unul de tip DCE. El permite utilizarea semnalelor de dialog pentru controlul fluxului de date conform protocolului RTS/CTS.

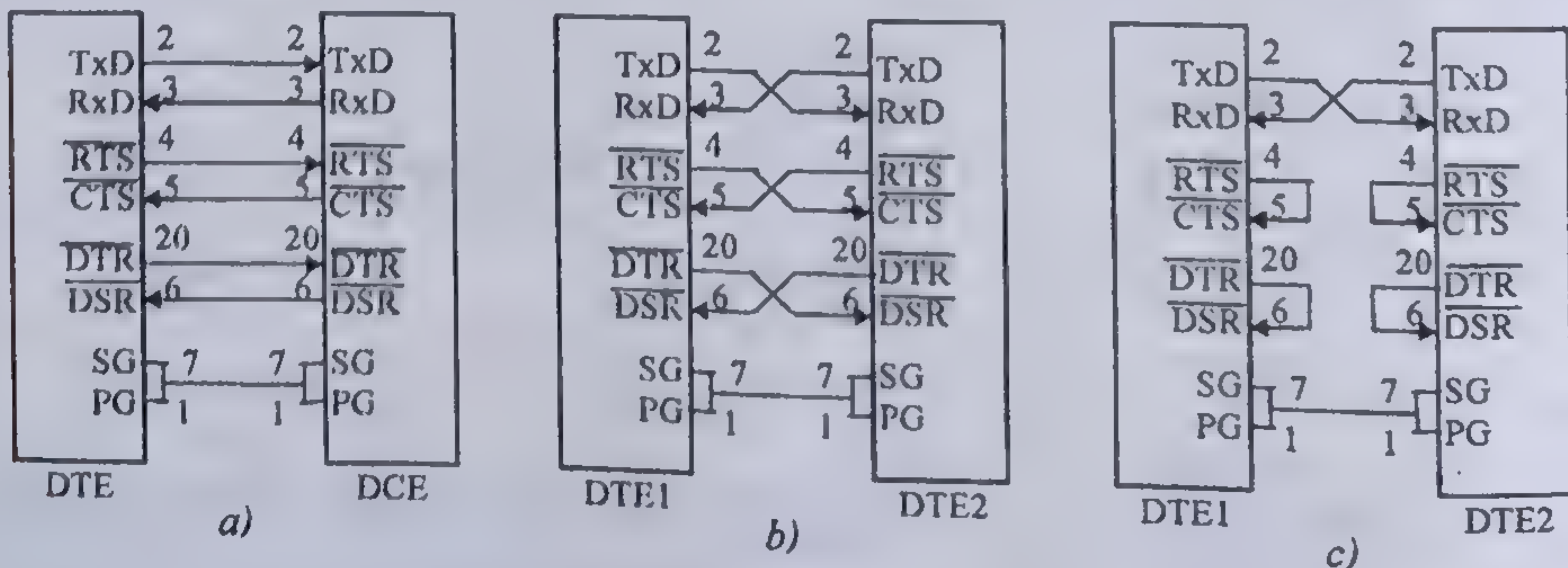


Fig.3.10. Conexiuni între echipamente compatibile RS-232 (DB-25P)

Interconectarea directă a două echipamente de tip DTE, fără modemuri, se utilizează pentru comunicație la distanțe mici (zeci de metri) și se bazează pe aceleași semnale de dialog (fig.3.10b). Pentru sincronizarea fluxului de date poate fi folosit același protocol, RTS/CTS.

În cazul unor distanțe mai mici (sub 30 metri), se poate folosi o interconectare simplificată, doar prin 3 fire, așa cum se prezintă în fig.3.10c. Liniile de dialog cu modemul se conectează în buclă locală, pentru a asigura îndeplinirea automată a condițiilor impuse de protocolul RTS/CTS. În acest caz, pentru sincronizarea fluxului de date dintre DTE1 și DTE2 trebuie să se folosească un protocol exclusiv software, cum este de exemplu protocolul XON/XOFF.

3.2.1.3. Protocolul XON/XOFF

Pentru controlul fluxului de date prin program se utilizează două cuvinte de comandă din setul de cuvinte de cod folosit pentru reprezentarea datelor, care se transmit de către dispozitivul care primește datele (receptor) către dispozitivul care le transmite (emițător).

Receptorul are prevăzută o zonă de memorie tampon, în care datele sunt depuse în ordinea în care sosesc pe linie. În cazul în care viteza cu care datele sunt preluate și utilizate este mai mică decât ritmul cu care acestea sunt depuse în zona tampon, există o tendință naturală de umplere a bufferului.

Înainte ca acest lucru să se întâmple, atunci când se atinge o limită de umplere superioară, de exemplu 75%, receptorul cere oprirea temporară a fluxului de date, prin transmiterea către emițător a caracterului XOFF (13h în cod ASCII). Se observă existența unei marje de siguranță, în acest caz de 25%, necesară pentru preluarea datelor care vor mai continua să sosească pe linie până în momentul în care emițătorul va recepționa caracterul XOFF și va opri efectiv transmisia.

Un timp, receptorul va prelua datele din zona tampon fără a primi altele noi, ceea ce va conduce la golirea treptată a acesteia. Când se ajunge la o limită inferioară de umplere a zonei tampon, de exemplu 25%, se transmite un caracter XON (11h în cod ASCII) către emițător, care îl informează că poate relua transmiterea datelor, după care procedura se repetă.

3.2.2. Standardele RS-423A și RS-422A

Datorită fiabilității relativ reduse a comunicației DTE/DTE bazate pe standardul RS-232, care poate asigura o transmisie sigură la 19200 Baud doar pe o distanță de cca. 20m, pentru distanțe mai mari au apărut noi standarde, care cresc performanțele liniilor de comunicație în mod semnificativ.

Standardul EIA RS-423A utilizează linii de joasă impedanță, de tip cablu coaxial de 50Ω, conectate la masă prin rezistoare, la capătul dinspre receptor,

pentru a preveni reflexiile. În fig.3.11 se prezintă modul de implementare a acestui standard folosind circuitele de adaptare electrică MC3487 (emițător) și MC3486 (receptor). Nivelul logic H este reprezentat de o

tensiune de -6÷-4V, iar nivelul logic L de o tensiune de +4÷+6V, ambele măsurate față de masă.

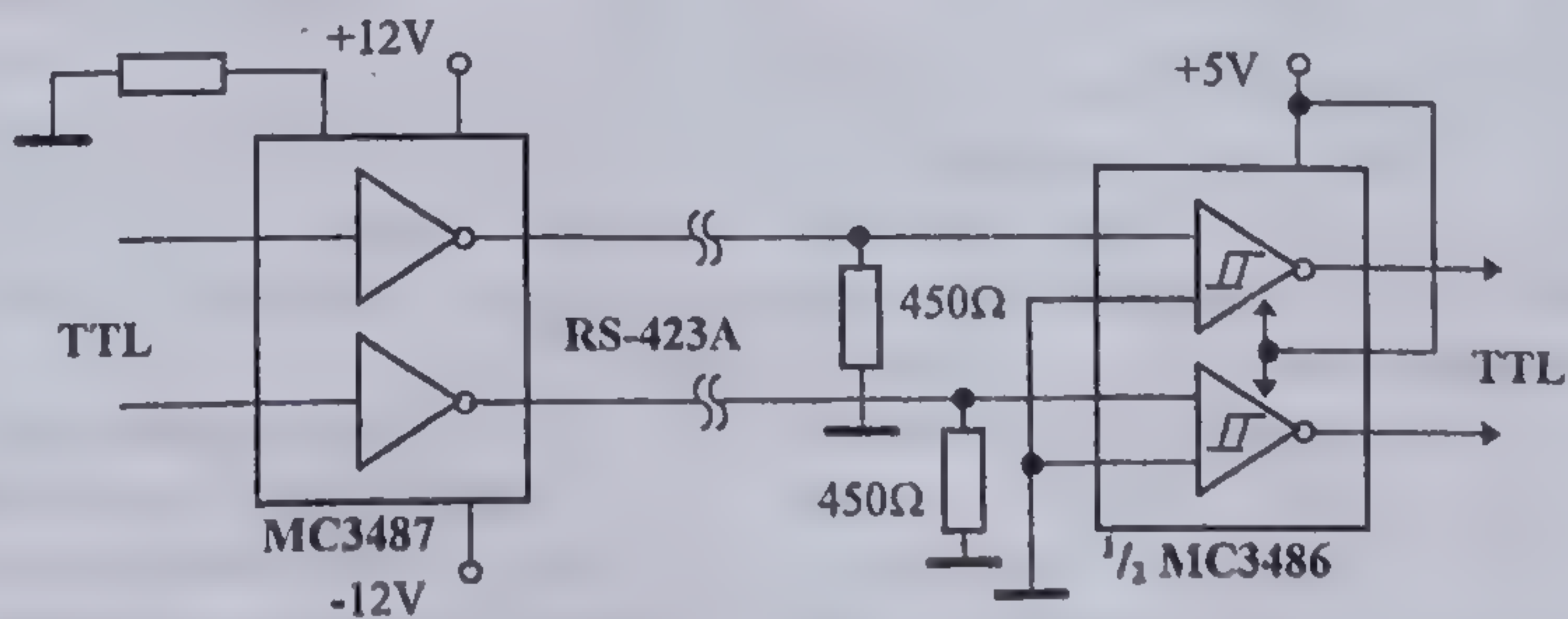


Fig.3.11. Conexiune RS-423A folosind circuitele MC3487 și MC3486

Standardul RS-423A permite utilizarea unei viteze de 100KBaud pe o linie de maximum 15m sau de 1200Baud pe o distanță de maximum 150m.

Standardul EIA RS-422A utilizează semnale diferențiale, transmise prin 2 fire *a* și *b*, sub formă de cablu panglică sau prin două fire torsadate (twisted pair). Noțiunea de semnal diferențial se referă aici la faptul că un nivel logic H se obține atunci când linia *b* este la un potențial mai ridicat decât linia *a*, iar situația opusă corespunde nivelului logic L. Diferența de potențial dintre cele două linii trebuie să fie de cel puțin 0,4V, dar nu mai mare de 12V. Spre exemplu, emițătorul de linie de tip MC3487, conectat ca în fig.3.12, generează o tensiune diferențială de cca. 2V. Tensiunea de mod comun trebuie să fie cuprinsă între -7 și +7V.

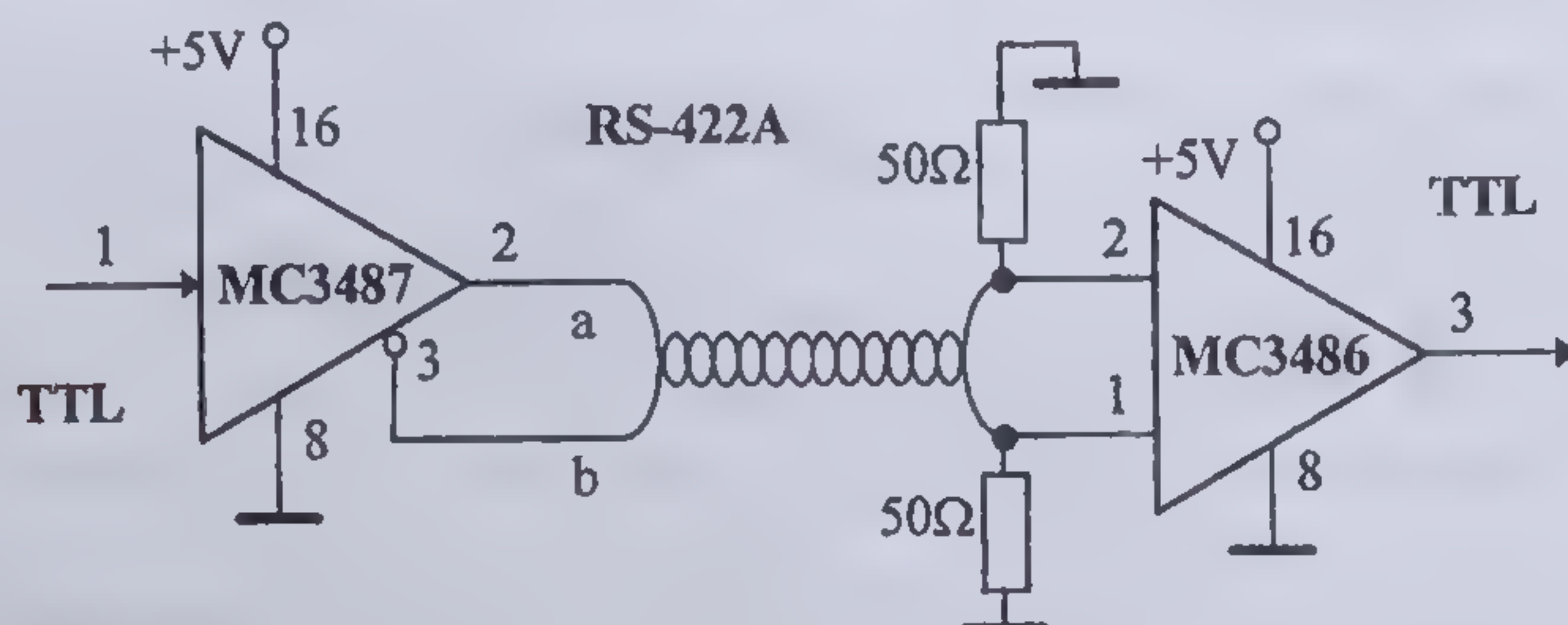


Fig.3.12. Conexiune RS-422A folosind circuitele MC3487 și MC3486

Pentru eliminarea reflexiilor, între liniile *a* și *b* se conectează un rezistor de valoare egală cu impedanța caracteristică a liniei, Z_0 (de obicei de 100Ω) sau, pentru o mai bună echilibrare a celor două semnale, fiecare linie se conectează la masă printr-un rezistor de valoare $Z_0/2$ (50Ω). Transmisia diferențială mai are și avantajul că perturbațiile care intervin acționează de aceeași manieră asupra ambelor linii, *a* și *b*; de aceea, ele se regăsesc în mare parte în tensiunea de mod comun și nu în cea de mod diferențial, care este purtătoarea de informație. Se pot obține astfel viteze maxime de comunicație de 10MBaud pe distanțe de până la 15m, respectiv de 100KBaud pe distanțe de maximum 150 m.

Standardele RS-423A și RS-422A nu specifică celelalte caracteristici ale legăturii seriale. Acestea sunt date de un standard separat, RS-449, care stabilește configurația a 37 de semnale pentru conectorul principal și a 9 semnale auxiliare pentru un conector opțional, în fapt un superset al semnalelor definite de standardul RS-232C.

3.2.3. Standardul RS-485

Ca și standardul RS-422A, acesta transmite semnal diferențial pe două fire torsadate (twisted pair), A și B. Însă, spre deosebire de standardul RS-422A, care este destinat legăturilor punct la punct, RS-485 permite realizarea de legături multipunct, prin conectarea la aceeași linie a mai multor sisteme (maximum 32). Se pot organiza astfel, la costuri relativ scăzute, rețele locale (fig.3.13) pentru comunicație în sisteme cu microprocesoare, destinate rulării de aplicații distribuite, în special în mediul industrial. Sistemele bazate pe microcontrolere au fost orientate cu predilecție pentru comunicația în standardul RS-485.

Ca interfață de linie pot fi folosite fie circuite integrate de tip emițător (SN75174) sau receptor (SN75175), fie circuite mixte, care conțin pe același cip atât un emițător cât și un receptor (SN75176). Ieșirile emițătorului și intrările receptorului se conectează între ele (la SN75176 sunt conectate intern) și formează un port diferențial conectat la cele două linii, A și B. Interfețele de linie, care constituie un nod al rețelei, pot să încarce magistrala cu una sau mai multe sarcini convenționale RS-485.

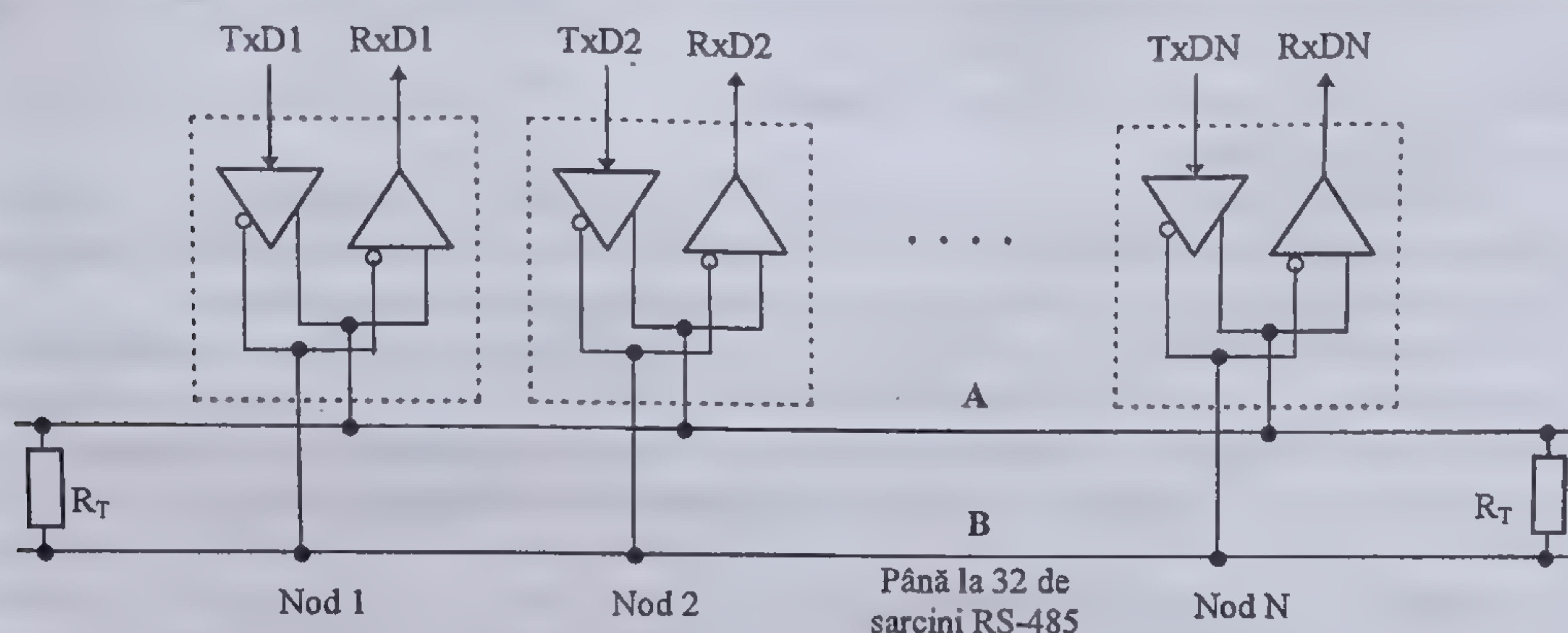
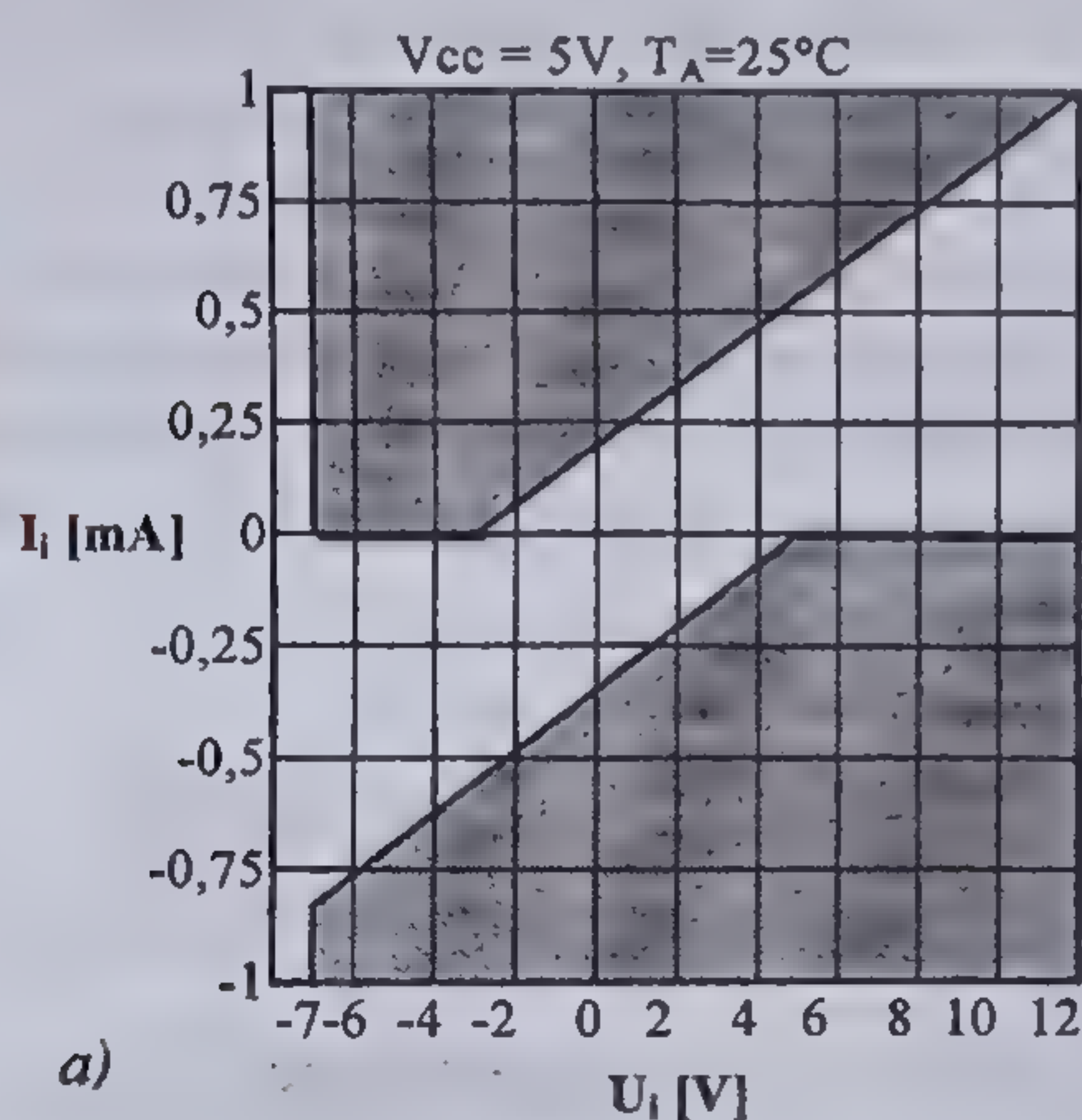
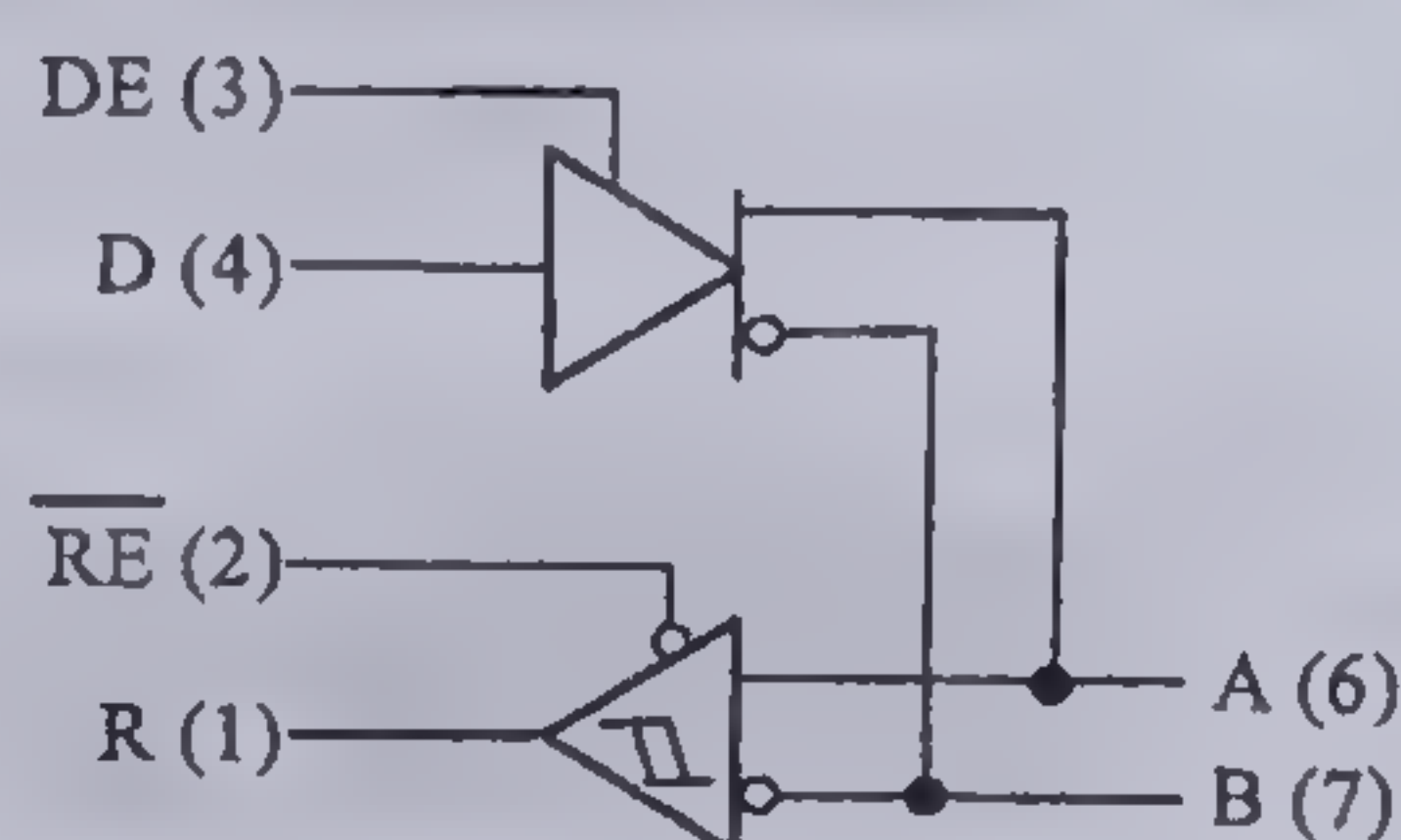


Fig.3.13. Rețea locală în standardul RS-485

În fig.3.14a este prezentată caracteristica de intrare curent/tensiune (I_i-U_i) corespunzătoare standardului, iar în figurile 3.14b și 3.14c schema logică și tabelul de adevăr pentru emițătorul și receptorul de linie al circuitului SN75176.



a)



b)

c)

Intrare TTL	Validare D	Ieșiri RS-485
D	DE	A B
0	1	L H
1	1	H L
0	0	HZ HZ
1	0	HZ HZ

Intrări RS-485	Validare R	Ieșire TTL
$V_{ID} = A-B$	\overline{RE}	R
$V_{ID} \geq 0,2V$	0	1
$-0,2V < V_{ID} < 0,2V$	0	interzis
$V_{ID} < -0,2V$	0	0
X	1	HZ

Fig.3.14. Caracteristica I_i-U_i a standardului RS-485 (a) și circuitul emițător/receptor de linie SN75176, specific acestui standard (b, c)

Porțiunea nehașurată din caracteristica I_i-U_i constituie zona permisă pentru evoluția semnalelor în standardul RS-485. Gama permisă a tensiunilor de intrare este între +12V și -7V, iar cele trei domenii logice (H, L și nedefinit) sunt determinate de tensiunea diferențială, $V_{ID} = 200mV$, între cele două linii de date, A și B. Sarcina unitate RS-485 (unitatea de încărcare a magistralei)

este de maximum 1mA și trebuie respectată de intrările receptoarelor. Pentru emisie, standardul RS-485 suportă 32 de sarcini și necesită, pentru eliminarea reflexiilor, terminatori de o valoare cât mai apropiată de impedanța caracteristică a magistralei ($R_T = 120\Omega$, montate între liniile A și B, la fiecare capăt al magistralei).

Emițătorul asigură la ieșire un curent de 60mA, iar intrările receptoarelor au o impedanță de intrare de $12K\Omega$ și o sensibilitate de $\pm 200mV$. Cablurile de legătură trebuie să aibă o impedanță caracteristică de minimum 120Ω , iar atenuarea să permită viteza dorită de comunicație.

În cazul în care se utilizează numai un singur nod emițător și unul sau mai multe noduri receptoare, liniile A și B trebuie prevăzute cu rezistoare "pull-up", la +5V (A) și respectiv la masă (B), de $330\div 470\Omega$. Lungimea maximă a liniei este de 2Km, dar la viteze reduse de comunicație. Pentru distanțe de ordinul zecilor de metri se pot utiliza viteze în jurul a 62,5 Kbaud.

3.3. Tipuri de comunicație serială

Principalul criteriu de clasificare a comunicațiilor seriale este modul în care se realizează sincronizarea transferului de date, atât la nivel fizic, cât și la cel al legăturii de date. La nivel fizic se utilizează două tipuri de comunicație: asincronă și sincronă, implementate în interfața serială (v. §2.2). La nivelul legăturii de date se utilizează protocoale asincrone sau sincrone, implementate prin software și parțial prin hardware (numai cele sincrone), cu ajutorul unor circuite specializate.

3.3.1. Comunicația serială asincronă (START-STOP)

Așa cum s-a menționat anterior, comunicația serială asincronă se caracterizează printr-o sincronizare la nivel de caracter. Datele se transmit pe linie sub formă de cuvinte de cod de 5÷8 biți, bit cu bit, începând cu bitul cel mai puțin semnificativ. Fiecare cuvânt de cod este precedat de un bit de START ("0" logic sau SPACE) și este urmat de 1, $1\frac{1}{2}$ sau 2 biți de STOP ("1" logic sau

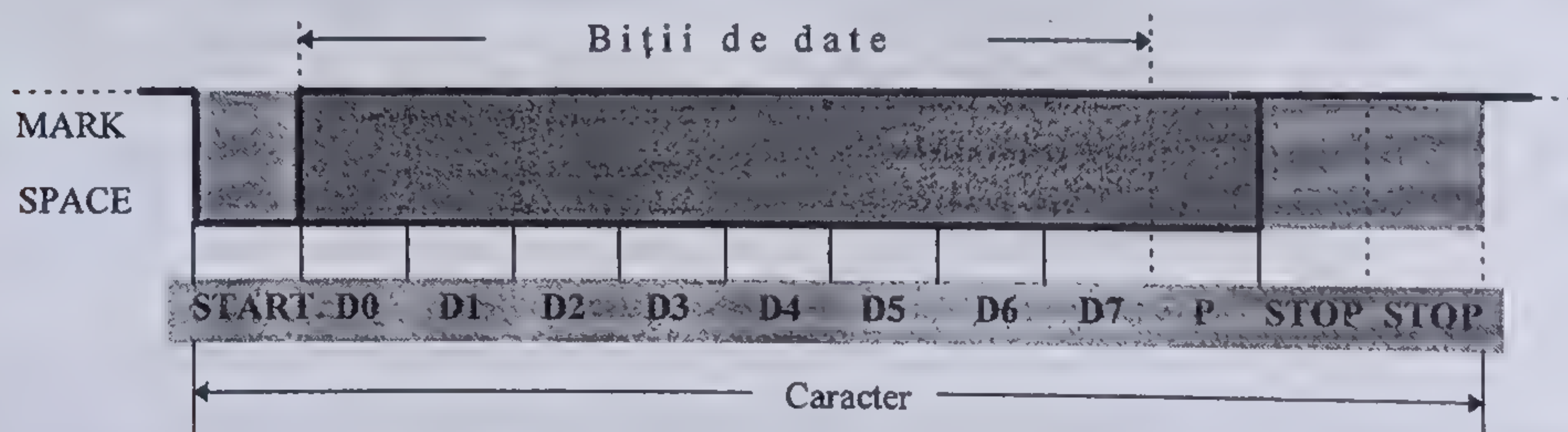


Fig.3.15. Formatul asincron al unui caracter pe linia serială

MARK), așa cum se prezintă în figura 3.15. Din acest motiv, comunicația de acest tip mai poartă numele de comunicație START-STOP. Acești biți au rolul de sincronizare a transferului la nivel de caracter. Opțional, între ultimul bit de date (cel mai semnificativ) și bitul (sau biții) de STOP, se inserează un bit de paritate, pentru detecția la recepție a erorilor prin metoda VRC (v. §3.1.4.1). Toate aceste informații constituie *formatul asincron* al unui caracter, așa cum apare el pe linia de comunicație serială.

Numărul de biți de date dintr-un caracter, utilizarea sau nu a bitului de paritate și tipul de paritate folosit (pară sau impară), precum și numărul de biți de STOP, sunt parametri care pot fi stabiliți la programarea circuitelor de interfață serială pentru comunicația asincronă. Pentru exemplificare, în fig.3.16 este prezentată evoluția în timp a semnalului pe linia TxD la transmisia

cuvântului de cod 8Bh, în cazul în care se lucrează cu 8 biți/caracter, paritate pară și 1 bit de STOP.

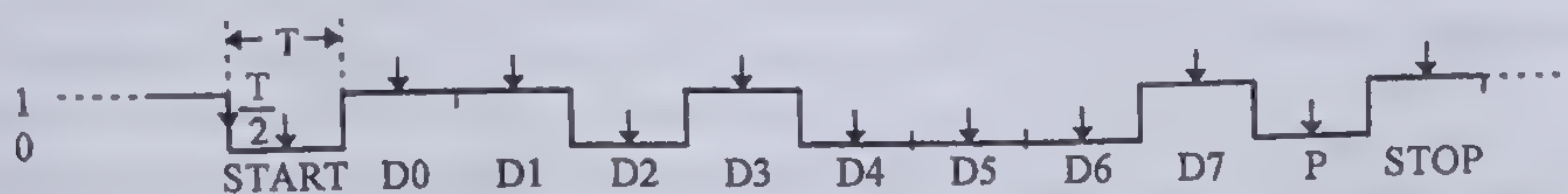


Fig.3.16. Formatul asincron al caracterului 10001011B (8Bh)
(8 biți/caracter, paritate pară, 1 bit de STOP)

La transmisia unui caracter se depune mai întâi pe linia TxD bitul de START, după care urmează biții de date, începând cu cel mai puțin semnificativ, bitul de paritate (opțional) și apoi bitul (biții) de STOP, care marchează sfârșitul caracterului. Fiecare bit este menținut pe linie un interval de timp egal cu perioada T, care stabilește astfel viteza de comunicație. În intervalul de timp care poate apare între sfârșitul transmisiei unui caracter și începutul transmisiei caracterului următor, linia TxD este menținută în starea MARK.

Recepția unui caracter se declanșează pe frontul descrescător al bitului de START. După un interval de timp egal cu $T/2$, receptorul verifică din nou existența bitului de START (nivel "0" logic), având astfel posibilitatea de a ignora tranzițiile false ale semnalului pe linie, datorate unor perturbații. Dacă se confirmă existența unui bit de START, receptorul pornește o bază de timp locală, care precizează, la fiecare perioadă T, momentele în care pot fi citiți următorii biți, marcate în fig.3.16 cu simbolul "↓".

Pentru ca transferul să poată avea loc, atât transmitătorul cât și receptorul trebuie să fie programați să lucreze cu aceiași parametri: viteza de comunicație (egală cu $1/T$), numărul de biți de date/caracter, condițiile privind utilizarea parității, numărul de biți de STOP.

Erorile care pot perturba transmisia serială asincronă sunt detectate și pot fi corectate la recepție în următoarele trei situații posibile:

- bitul de paritate adăugat de transmitător la sfârșitul fiecărui caracter nu coincide cu bitul de paritate recalculat de receptor - *eroare de paritate*;
- nerecepționarea unui bit de STOP, care să marcheze corect sfârșitul caracterului - *eroare de încadrare*;
- receptorul a primit în registrul de deplasare serie/paralel un caracter pe care l-a transferat în registrul tampon de recepție peste caracterul precedent, nepreluat la timp de către microprocesor - *eroare de suprapunere*.

Comunicația de tip asincron folosește viteze relativ mici, atât pentru legăturile de date prin modemuri, cât și pentru legăturile directe în curent continuu, fără modem (rate de până la 19200 Baud). Ea este eficientă pentru vehicularea unui volum redus de date, compus din mesaje scurte. Datorită faptului că transmisia asincronă este o transmisie în mod caracter, este specifică terminalelor cu interacțiune directă cu operatorul uman. Numărul de biți vehiculați pe linie pentru N caractere variază între $(START+n+STOP)N = (n+2)N$ și $(START+n+P+2 \times STOP)N = (n+4)N$, unde n e numărul de biți de date al unui caracter.

Eficiența comunicației asincrone se poate determina ca fiind raportul dintre numărul de biți de date și numărul total de biți transmiși pe linie în același interval de timp și poate evolua între $n/(n+4)$ și $n/(n+2)$, respectiv între valorile minimă de 0,55 și respectiv maximă de 0,8 (pentru $n=8$ biți/caracter, fără bit de paritate și un singur bit de STOP). Se observă că, deoarece blocul de date este constituit dintr-un singur caracter, eficiența comunicației asincrone nu depinde de numărul de caractere transferat, ci doar de parametrii comunicației seriale.

3.3.2. Comunicația serială sincronă

Spre deosebire de comunicația asincronă, comunicația sincronă se realizează la nivel de mesaj. Datele sunt transferate sub forma unor șiruri de biți (mesaje) de lungime variabilă, fără biți de START și STOP între cuvintele de cod. În acest caz, un cuvânt de cod este echivalent cu un caracter. Sincronizarea se face la nivelul de mesaj, prin încadrarea sa între caractere speciale de sincronizare, delimitare și de comandă.

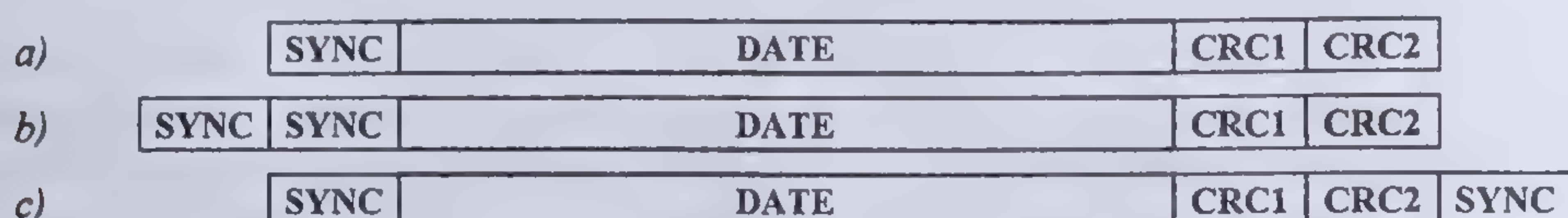


Fig. 3.17. Formate ale mesajelor specifice transmisiilor sincrone

Astfel, transmisia sincronă începe cu unul (fig.3.17a) sau două (fig.3.17b) caractere de sincronizare (SYNC) și continuă cu transmiterea blocului de date, care poate conține alte caractere de sincronizare precum și un număr variabil de caractere de date și de comandă (de obicei între 10^2 și 10^5). În final sunt adăugate unul sau două caractere pentru detecția erorilor (CRC1, CRC2). Pentru a se menține sincronizarea pe durata transmiterii unui mesaj se utilizează inserarea automată de caractere de sincronizare în șirul de date, la un interval de 1-2 secunde. Între două mesaje succesive, linia TxD este menținută în starea MARK (fig.3.17a și b) sau se pot transmite continuu caractere de sincronizare (fig.3.17c).

Datorită faptului că între ETCD (modem) și ETPD (interfața serială) trebuie menținut sincronismul pe un interval de timp mult mai mare decât în cazul comunicației asincrone, cele două echipamente trebuie să lucreze cu un semnal de tact comun. Baza de timp comună poate să provină fie dintr-o sursă de semnal externă, prevăzută la nivelul modemului (vezi fig.3.7), sau este extrasă din datele recepționate de modem cu ajutorul unei bucle de calare pe fază (Phase Locked Loop - PLL). În cazul utilizării unei bucle PLL, modemul receptor își ajustează permanent frecvența semnalului de tact după cea a transmițătorului, în funcție de schimbările de fază detectate în semnalul recepționat. Altfel spus, tranzițiile existente în semnalul recepționat sunt folosite de către receptor pentru a se menține în sincronism cu transmițătorul. Aceste tranziții lipsesc atunci când se transmite un șir lung de biți de același tip. De aceea, se utilizează metode speciale de codificare (NRZI, Manchester etc.), care să garanteze existența unor tranziții ale semnalului transmis pe linie, chiar și atunci când acestea lipsesc în datele care au fost codificate.

Transmisia sincronă se utilizează pe legături de date de viteză mare, eficiente, între sisteme cu microprocesoare, motiv pentru care stă la baza majorității protocoalelor de comunicație standard.

3.3.3. Protocoale de date

Așa cum am arătat în §3.1.1, la nivelul legăturii de date se utilizează protocoale de tip DLC (Data Link Control). Protocoalele în care funcția de control a legăturii de date este îndeplinită de un set de caractere speciale, numite caractere de control, se numesc *protocoale orientate pe caracter* (Byte Control Protocols - BCP). Protocoalele în care controlul legăturii de date se face prin utilizarea uneia sau a cel mult două secvențe de biți se numesc *protocoale orientate pe bit* (Bit Oriented Protocols - BOP). Deși ele au fost concepute în special pentru comunicației sincrone, principiile care stau la baza lor pot fi aplicate cu succes și în contextul comunicației asincrone.

3.3.3.1. *Protocoale de comunicație orientate pe caracter (BCP)*

În acest caz, un mesaj se transmite sub formă de blocuri de date numite blocuri BCP. Ele sunt compuse dintr-un câmp de comandă (antet sau "header"), un câmp de text (datele propriu-zise) și un câmp de verificare a erorilor sau terminator de bloc, așa cum se prezintă în fig.3.18.

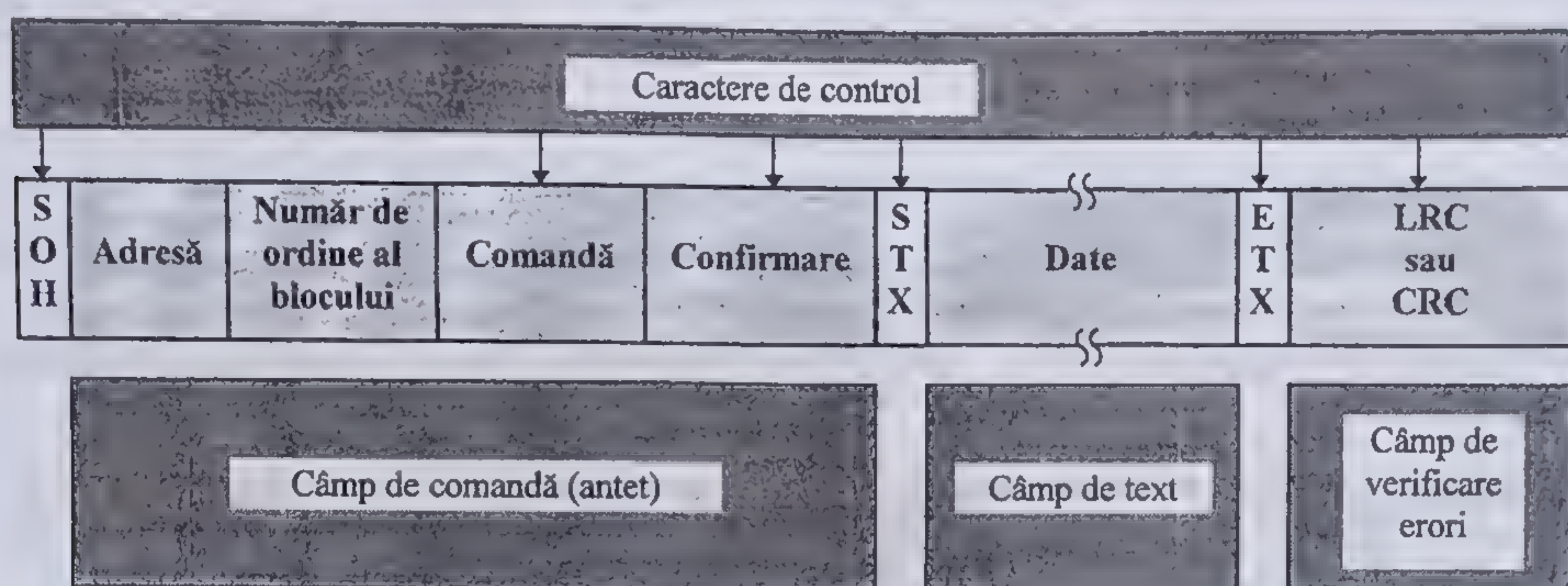


Fig.3.18. Formatul unui bloc BCP

Drept delimitatori de câmp sau pentru comenzi și confirmări se folosesc caractere de control specifice codului utilizat pentru reprezentarea informației (ASCII, EBCDIC etc.). Spre exemplu, în cadrul codului ASCII este prevăzut un subset de caractere de control, din care o parte sunt destinate transmisiilor de date (tab.3.1).

Tabelul 3.1.

Caracter de control	Valoare (hexa)	Semnificație	Mod de utilizare
SOH	01	Start Of Header	marchează începutul câmpului antet
STX	02	Start Of Text	marchează începutul câmpului text
EOT	03	End Of Transmission	terminarea normală a legăturii de date
ENQ	04	ENQuiry	cere stabilirea unei legături de date
ACK	05	ACKnowledge	confirmă recepția fără erori a unui bloc
NACK	06	Negative ACKnowledge	infirmare (au apărut erori)
SYN	16	SYNchronization	caracter de sincronizare (SYNC)
ETB	17	End Transmit Block	indică terminarea transmisiei unui bloc
CAN	18	CANcel	indică abandonarea transmisiei mesajului curent

La transmisie, un mesaj este împărțit în mai multe blocuri numerotate. Antetul unui bloc BCP conține informații auxiliare privind adresa sistemului sursă sau destinație, tipul blocului (date, comandă sau confirmare), precum și numărul blocului în cadrul mesajului, dacă este un bloc de date.

Câmpul de text apare numai în blocurile de date și poate să conțină caractere ale codului folosit sau caractere și secvențe de caractere *transparente* față de acest cod. Problema transparenței se pune în cazul transmisiei de date binare și este foarte importantă, deoarece caracterele de control se pune în cazul transmisiei de date binare și este foarte importantă, deoarece caracterele de control utilizate de protocol nu trebuie să apară în câmpul de text. În caz contrar, ele vor fi interpretate la recepție implicit drept cuvinte de control și nu drept date. Implementarea unui mod de transmisie transparent este specifică fiecărui protocol în parte.

Asigurarea integrității unui mesaj se face prin asigurarea integrității fiecărui bloc de date din care este compus. În acest scop, fiecare caracter din bloc poate să conțină informații de detectare a erorilor, așa cum este cazul bitului de paritate adăugat la sfârșit de caracter (VRC). În

plus, fiecare bloc conține astfel de informații în câmpul de verificare a erorilor. Ele pot fi de tip LRC - adică suma modulo 2 a tuturor caracterelor din bloc cu excepția celor de control și a informației transmisă transparent, sau de tip CRC, dacă se utilizează coduri cu redundanță ciclică (v. §3.1.4.).

Cele mai utilizate protocoale standard de tip BCP sunt MONOSYNC (un singur caracter de sincronizare), BISYNC (IBM) și DDCMP (Digital Data Communications Message Protocol, al firmei DEC).

3.3.3.2. Protocoale de comunicație orientate pe bit (BOP)

Protocoalele de tip BOP folosesc mesaje formate din cadre (frames). Acestea sunt blocuri de date cu structura din fig.3.19, deosebirea principală față de blocurile BCP constând în aceea că ele pot asigura funcționarea corectă a legăturii de date cu numai unul sau două caractere specifice de control.

Delimi-tator	Adresă	Comandă	Informație	Verificare erori	Delimi-tator
01111110	1 sau mai multe caractere de 8 biți	1 sau 2 caractere de 8 biți	0 sau mai multe caractere (oricâte pentru ADCCP/HDLC, multiplu de 8 biți pentru SDLC)	Rest CRC: CRC-16 sau CCITT	01111110

Fig. 3.19. Formatul unui cadru BOP

Ca exemple de protocoale standard BOP pot fi enumerate: SDLC (Synchronous Data Link Control - IBM), HDLC (High-Level Data Link Control - ISO), ADCCP (Advanced Data Communication Control Procedures - ANSI), BDLC (Burroughs' Data Link Control).

Delimitarea începutului și sfârșitului unui cadru se face cu ajutorul unui singur caracter, numit *delimitator* sau FLAG, care are practic un rol similar cu cel al caracterului de sincronizare, SYN, din protocoalele BCP. Câmpurile care urmează nu mai sunt separate de caractere de control, ca în protocoalele BCP, ci ele sunt ușor de identificat datorită poziției lor față de începutul cadrului. Astfel, câmpurile de adresă, de comandă și de verificare a erorilor au fiecare o lungime fixă, în timp ce câmpul de informație are o lungime variabilă.

Aceste protocoale sunt adecvate implementării comunicației de date multipunct de tip master/slave, în configurații de tip rețea (vezi §3.1.3.1).

Sistemul coordonator (master) transmite un cadru către un sistem slave, după ce în prealabil i-a precizat *adresa* în câmpul corespunzător. Sistemele subordonate (slaves) caută în permanentă pe linie secvența *delimitator-adresă*. Dacă adresa proprie coincide cu câmpul adresă recepționat, sistemul slave validează recepția cadrului, altfel intră din nou în așteptarea secvenței *delimitator-adresă*. După recepția completă a cadrului (detectarea delimitatorului de sfârșit de cadru) se verifică corectitudinea mesajului. Dacă datele sunt eronate și cadrul trebuie retransmis, sistemul slave transmite un cadru de înfirmare către master. Dacă datele recepționate sunt corecte și necesită un răspuns din partea sistemului slave, acesta pregătește și transmite unul sau mai multe cadre de răspuns către master. Orice cadru transmis de un sistem slave poartă, în câmpul special prevăzut în acest scop, adresa sistemului slave respectiv.

Câmpul de *comandă* determină tipul cadrului și numărul de ordine al acestuia în contextul mesajului. Sistemul master îl folosește pentru a indica sistemului slave ce operație trebuie să execute, iar sistemul slave pentru a răspunde despre modul în care a realizat operația solicitată de master.

Câmpul de *informație* are o lungime variabilă și poate conține date în orice cod binar. Problema transparenței transmisiei se reduce la faptul că între cei doi delimitatori ar putea să apară o succesiune de 6 biți de "1" încadrați de biți de "0", care ar fi interpretată în mod prematur ca fiind un delimitator de cadru. Pentru a elimina această posibilitate, după emisia delimitatorului de început de cadru, sistemul transmițător numără biții succesivi de "1" și inserează în mod automat, după cel de-al 5-lea bit "1", un bit "0" (excepție face delimitatorul de sfârșit de cadru), așa cum reiese din primele două diagrame din fig.3.20.

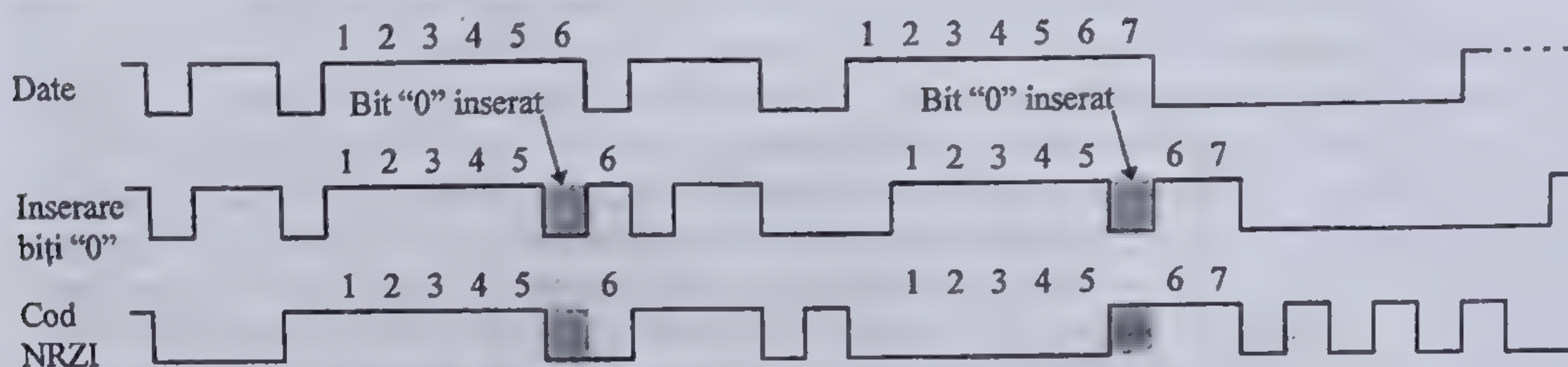


Fig.3.20. Codificarea NRZI, precedată de inserarea de zerouri

În cazul în care receptorul primește 5 biți succesivi de "1", dacă bitul imediat următor este "0", acesta se elimină în mod automat și se continuă recepția, iar dacă este "1" atunci se consideră că s-a detectat un delimitator de început sau de sfârșit de cadru.

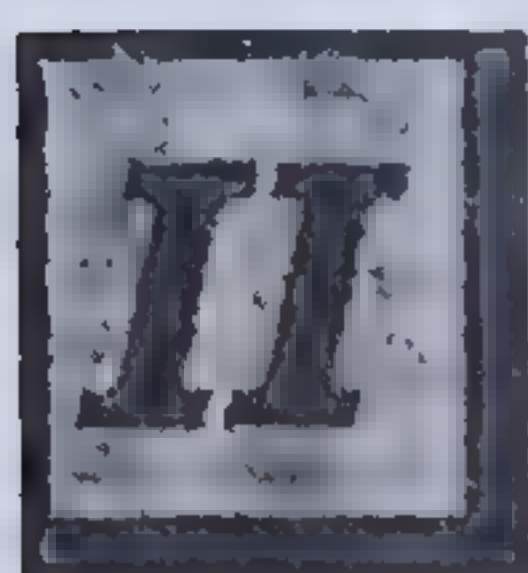
Inserarea automată de zerouri în semnalul transmis pe linie rezolvă parțial și problema păstrării sincronismului între transmițător și receptor, deoarece limitează lungimea șirurilor de biți "1" la 5. Rămâne de rezolvat cealaltă situație: limitarea șirurilor lungi de biți "0". O metodă frecvent utilizată (cazul protocolului SDLC/HDLC) este cea cunoscută sub numele NRZI (Non Return to Zero Inverted). Aceasta prevede că starea semnalului transmis pe linie nu se schimbă față de momentul anterior dacă trebuie transmis un bit "1", dar se schimbă dacă trebuie transmis un bit "0", așa cum se reiese din ultima diagramă din fig.3.20. Se observă că semnalul transmis pe linie poate avea cel mult 6 biți "1", respectiv cel mult 6 biți "0" succesivi.

La recepție se realizează mai întâi decodificarea NRZI și apoi eliminarea zerourilor inserate, pentru păstrarea transparenței. O excepție o constituie transmisia unei secvențe ABORT, care constă dintr-un șir de cel puțin 7 biți "1" consecutivi. Prezența unei astfel de secvențe pe linie indică receptorului că trebuie să abandoneze recepția cadrului curent și să intre în așteptarea următorului cadru. Secvența delimitator și secvența ABORT sunt cele două secvențe care realizează sincronizarea în comunicația sincronă de tip BOP.

Modul în care se aplică metodele de păstrare a transparenței și a sincronismului scoate în evidență cu claritate faptul că mai importantă decât împărțirea biților pe caractere este ordinea de succesiune a biților într-un cadru, indiferent de caracterele din care fac parte.

Câmpul de verificare a erorilor se completează cu restul împărțirii polinomului asociat șirului de biți ai unui cadru (cu excepția celor doi delimitatori și a biților "0" inserați pentru păstrarea transparenței) la un polinom generator. În acest scop se utilizează, de regulă, polinoamele generatoare cunoscute sub numele CRC-16 ($g(X) = X^{16} + X^{12} + X^5 + 1$) sau CCITT ($g(X) = X^{16} + X^{15} + X^2 + 1$).

Toate operațiile suplimentare pe care le necesită funcționarea unui protocol BOP: codificare/ decodificare pentru asigurarea transparenței, inserție/eliminare zerouri, generare/control CRC, căutare delimitator, recunoaștere adresă etc. pot fi realizate de către circuite specializate, așa cum este, de exemplu, circuitul 8273 pentru protocoalele SDLC/HDLC.



SISTEME CU MICROPROCESOARE DE 8 BIȚI

După cei peste 25 de ani de la apariție, sistemele cu microprocesoare de 8 biți ocupă încă un segment important în aplicațiile din diferite domenii. Deși posedă o putere redusă de calcul în raport cu microprocesoarele de 16 sau 32 de biți, acestea sunt avantajoase, ca raport performanță/cost, la implementarea de echipamente numerice. De asemenea, sunt folosite în realizarea structurilor de conducere distribuite, unde se regăsesc pe nivelul inferior al ierarhiei. Îmbunătățirea succesivă a performanțelor, precum și dezvoltarea unor familii de dispozitive puternice în jurul lor, fac în continuare atractivă această categorie de microprocesoare pentru proiectanții de sisteme numerice. Un exemplu concludent este produsul 8080, pentru care Intel a dezvoltat o gamă variată de dispozitive LSI ce suplinesc multe din dezavantajele sale. În plus, apariția ulterioară a microprocesorului 8085 - 100% compatibil software în jos - determină ca acest prim produs să fie încă utilizat. Un al doilea exemplu îl constituie microprocesorul Z80, care a preluat o bună parte din concepția lui 8080 și care este cel mai apreciat procesor de 8 biți. Mai mult, pe baza arhitecturii microprocesoarelor de uz general s-au elaborat microcontrolerele actuale de 8 biți, cu facilități deosebite în ceea ce privește implementarea controlerelor industriale.

Practic, categoria microprocesoarelor de uz general este cea mai numeroasă, ceea ce poate fi explicat prin faptul că majoritatea firmelor producătoare de dispozitive numerice integrate au fost interesate și în realizarea de microprocesoare. În tabelul II.1 sunt prezentate date sintetice cu privire la principalii producători de microprocesoare de 8 biți și unele caracteristici ale acestora [11, 23, 43, 44]. Valoarea 2,5 din ultima coloană indică faptul că microprocesoarele respective sunt cele mai evolute produse din această clasă ale firmei respective.

Tabelul II.1.

Firma produ- cătoare	Denumire micro- procesor	Tehno- logie	Lun- gime cuvânt	Memorie max. direct adresabilă (octeți)	Număr ins- truc- țiuni	Număr de între- ruperi	Număr registre gene- rale	Nu- măr de pini	Tensi- uni de alimen- tare (V)	Ge- ne- ra- ția
INTEL	8008	PMOS	8/8	16k	48	1	6	18	+5; -9	1
INTEL	8080A	NMOS	8/8	64k	78	1	8	40	±5; +12	2
INTEL	8085	NMOS	8/8	64k	80	5	8	40	+5	2,5
Motorola	6800	NMOS	8/8	64k	72	1	0	40	+5	2
Motorola	6809	NMOS	8/8	64k	59	1	0	40	+5	2,5
ZILOG	Z80	NMOS	8/8	64k	158	2	14	40	+5	2,5
Fairchild	F8	NMOS	8/8	64k	69	1	64	40	+5; +12	2
RCA	CDP 1802	CMOS	8/8	64k	91	1	16	40	+3...+12	2
MOSTEK	3874	NMOS	8/8	64k	70	4	64	40	+5	2
Signetics	2650	NMOS	8/8	32k	75	1	7	40	+5	2
MOS Tech	6502	NMOS	8/8	64k	56	1	0	40	+5	2
National Semicond.	NSC800	CMOS	8/8	64k	150	15	14	40	+5	2

Numărul mare de produse din această categorie nu permite abordarea lor exhaustivă și de fapt nici nu ne-am propus acest lucru. Considerăm necesară prezentarea tipurilor de microprocesoare cel mai mult utilizate și cu o largă răspândire în țara noastră.



Sisteme cu microprocesoare Intel de 8 biți

Atât din dorința de a evidenția continuitatea în dezvoltarea sistemelor bazate pe microprocesoare, dar și din necesitatea abordării graduale a problematicii, considerăm că nu poate fi evitată această categorie de dispozitive. Mai mult, prin produsele sale, Intel a definit arhitectura generațiilor de microprocesoare care au urmat și, după cum s-a mai arătat, 8080 reprezintă un "standard" pentru microprocesoarele de 8 biți. Preluând avantajele acestuia, 8085 a adus noutăți de concepție și a prefigurat apariția sistemelor "împachetate" (embedded systems), orientate spre realizarea unor microsisteme cu număr redus de dispozitive LSI.

4.1. Sisteme bazate pe microprocesorul Intel 8080A

Mult timp acest tip de microprocesor a deținut topul în aplicații, atât în domeniul microcalculatoarelor, cât și în cel al controlului automat. Cel puțin în țara noastră, chiar până către sfârșitul deceniului opt, a ocupat cea mai mare pondere în aplicațiile sistemelor cu microprocesoare. Fiind intens studiat de către cercetători, este explicabilă și cantitatea mare de literatură apărută despre acest microprocesor în limba română [5, 11, 26, 31, 39].

Din acest motiv, vom prezenta pe scurt principalele sale caracteristici, detaliind acele aspecte care au o mai mare generalitate și privesc realizarea aplicațiilor de timp real: organizarea sistemelor de întreruperi, introducerea timpului ca variabilă independentă în prelucrarea informațiilor, accesul direct la memorie, comunicația interprocesor și interfațarea cu operatorul.

4.1.1. Microprocesorul 8080A. Caracteristici generale

Fabricat în tehnologie NMOS, conține aproximativ 2900 de tranzistoare convenționale încapsulate într-un dispozitiv cu 40 de pini. Principalele caracteristici ale microprocesorului [43] sunt:

- prelucrează cuvinte de 8 biți în binar și BCD;
- poate adresa o memorie externă de 64Kocteți;
- poate adresa 256 dispozitive de intrare și 256 dispozitive de ieșire;
- organizează stiva de lucru în memoria RAM externă;
- prezintă compatibilitate TTL la intrare și ieșire ($I_{IL} = 1,9 \text{ mA}$, dar $V_{ILmin} = 3,3 \text{ V}$), cu excepția semnalelor de tact, care sunt de nivel MOS;
- poate executa un set de 78 de instrucțiuni, la o frecvență de tact cuprinsă între 500KHz și 2MHz ($t_{CY} = 0,48 + 2\mu s$);
- are o singură intrare de întreruperi, mascabilă prin program;
- necesită trei surse de alimentare: +5V, -5V și +12V.

În fig.4.1 este prezentată schema bloc internă a microprocesorului 8080, din care se pot evidenția următoarele unități funcționale:

- unitatea logico-aritmetică (ULA);
- zona registrelor de date și adrese;
- blocul de decodificare a instrucțiunilor;
- blocul de secvențiere și comandă;
- magistrala internă de date, bidirecțională.

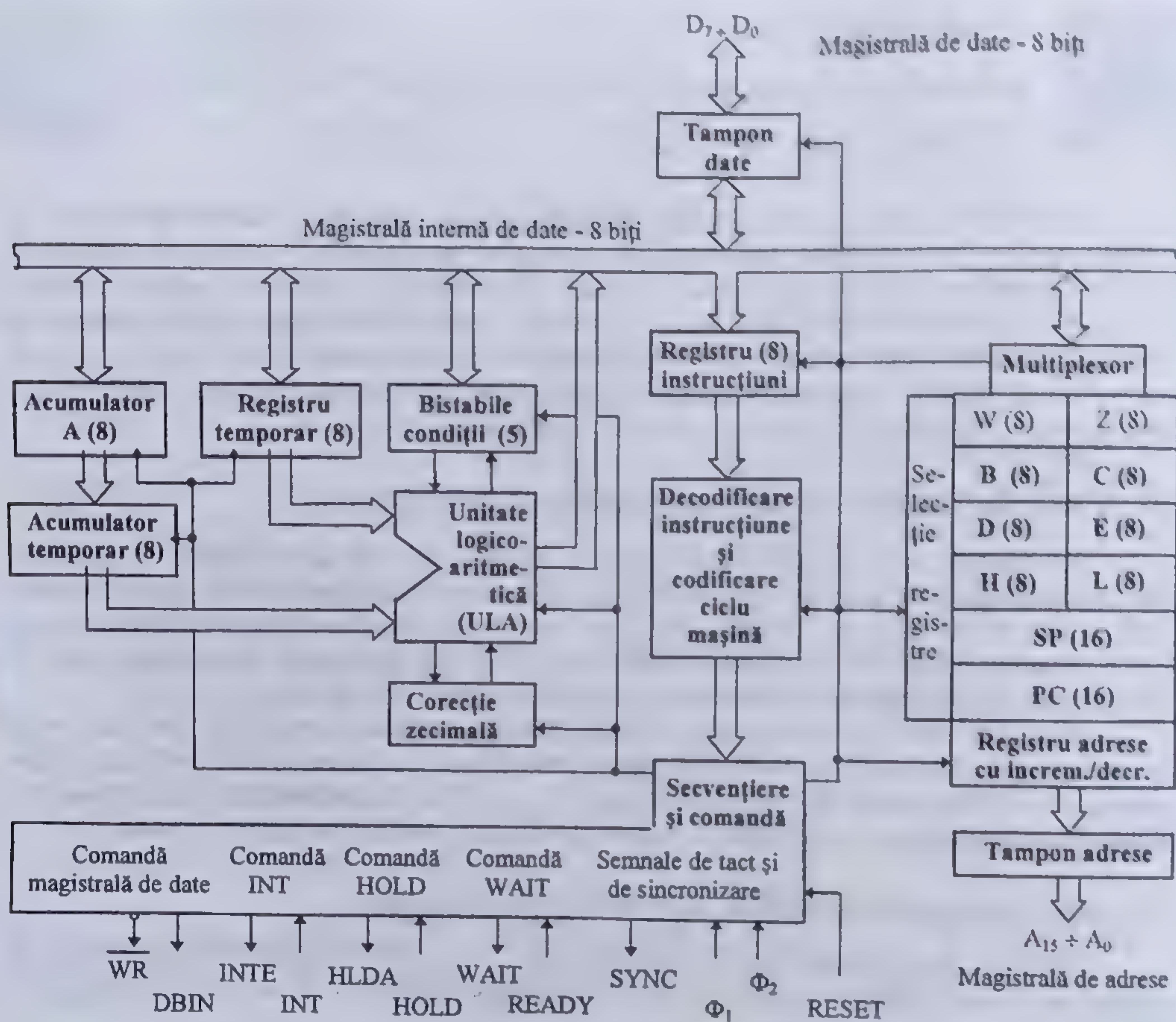


Fig.4.1. Schema bloc a microprocesorului 8080A

Unitatea logico-aritmetică (ULA) conține un sumator paralel de 8 biți și are atașate:

- un acumulator (A) de 8 biți, accesibil utilizatorului;
- un acumulator temporar de 8 biți, care nu este accesibil direct prin program și se folosește pentru memorarea primului operand;
- un registru temporar de 8 biți, de asemenea inaccesibil utilizatorului și care este folosit pentru memorarea celui de-al doilea operand;
- cinci bistabile de condiții: CY (Carry), AC (Auxiliary Carry), Z (Zero), S (Sign), P (Parity);
- un circuit de corecție automată pentru lucrul cu operanzi în BCD.

Bistabilele de condiție pot fi manipulate sub forma unui registru de 8 biți, care împreună cu conținutul acumulatorului formează *cuvântul de stare al programului* - PSW (Program Status Word), de 16 biți.

									D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
PSW	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀	S	Z	-	AC	-	P	-	CY

Zona registrelor formează o mică memorie RAM internă, compusă din 6 registre de câte 16 biți fiecare:

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

- contorul de program, PC (Program Counter);
- indicatorul vârfului stivei, SP (Stack Pointer);
- șase registre generale de lucru: B, C, D, E, H și L, ce pot fi adresate individual, ca registre de 8 biți sau în pereche, ca registre de 16 biți;
- un registru pereche W-Z, inaccesibil utilizatorului, ce este folosit de microprocesor ca registru temporar, la execuția internă a instrucțiunilor.

Conținutul registrului PC poate fi incrementat/decrementat și memorat temporar (latch) înainte de a fi transferat în registrul tampon (buffer) de adresă. Acesta din urmă depune adresa pe liniile $A_{15} \div A_0$ și asigură amplificarea lor la nivelul unei sarcini TTL.

Un registru poate fi citit sau înscris pe magistrala internă printr-un multiplexor, care este adresat prin intermediul unui circuit de selecție, sub controlul blocului de secvențiere și comandă. Magistrala internă de date este izolată de magistrala externă, $D_7 \div D_0$, prin intermediul unui registru tampon bidirecțional, care amplifică semnalele la nivelul unei sarcini TTL și asigură memorarea temporară a datelor transmise de microprocesor. Prin intermediul aceluiași registru tampon microprocesorul primește din memorie atât datele cât și codurile instrucțiunilor.

La începutul execuției unei instrucțiuni, codul acesteia este încărcat în *registrul de instrucțiuni*, de unde este preluat de *blocul de decodificare a instrucțiunilor*. Acest bloc conține o memorie ROM, programată la realizarea microprocesorului astfel încât să permită recunoașterea codurilor corespunzătoare setului de instrucțiuni cu care este dotat. După decodificarea instrucțiunii se realizează o codificare a ciclurilor mașină care trebuie realizate în etapa de execuție.

Blocul de secvențiere și comandă realizează desfășurarea temporală a ciclurilor mașină aferente execuției unei instrucțiuni, prin raportare la semnalele de tact primite de la un generator extern cu doua faze, Φ_1 și Φ_2 . Semnalele Φ_1 și Φ_2 trebuie să fie riguros sincronizate, să posede o anumită formă de undă pe durata unui ciclu, t_{CY} , iar nivelul "1" logic trebuie să fie de +12V (și nu mai puțin de +9V). Evoluția celor două semnale se raportează la perioada unui oscilator pilot astfel încât frecvența acestuia să respecte relația $f = 9/t_{CY} = 4,5 \div 18,432$ MHz.

Cronograma ideală pentru semnalele Φ_1 și Φ_2 este prezentată în fig.4.2, în raport cu cele 9 perioade ale oscilatorului pilot.

Firma Intel a realizat un dispozitiv integrat, cu codul 8224, care asigură generarea corectă a semnalelor de tact. Pentru sincronizarea cu exteriorul, microprocesorul 8080 generează un impuls SYNC la începutul fiecărui ciclu mașină, încadrat de fronturile ascendente ale lui Φ_2 (fig.4.2).

Pe durata semnalului SYNC, microprocesorul depune pe magistrala de date un *octet de stare* (Status Byte), care conține informațiile necesare despre tipul operației ce urmează a fi efectuată în ciclul mașină respectiv. Se utilizează o procedură de multiplexare în timp a informației, care permite folosirea liniilor $D_7 \div D_0$ pentru transmiterea de comenzi în exterior, urmată de revenirea la funcția de bază, aceea de linii de date. Cunoașterea acestui fapt este importantă pentru proiectantul de sistem, care trebuie să prevadă un registru cu zăvorăre (latch) în exterior pentru reținerea octetului de stare. Această tehnică conduce la o economie de conexiuni externe (pini) și a fost utilizată de majoritatea producătorilor de

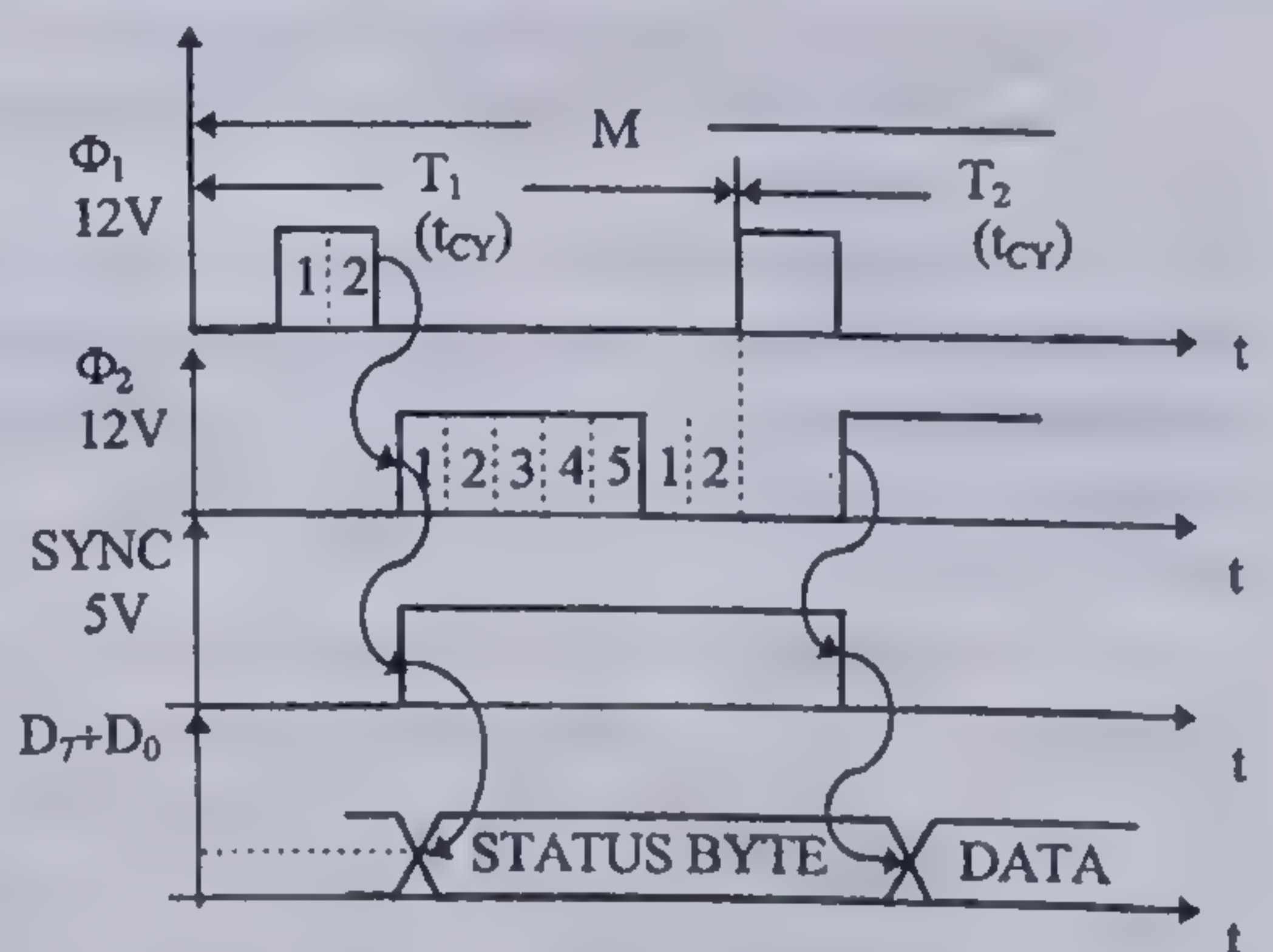


Fig.4.2. Diagrama pentru semnalele Φ_1 , Φ_2 și

microprocesoare. Tot pentru sincronizarea cu exteriorul, 8080 generează încă două semnale, având următoarele semnificații:

- DBIN (Data Bus INput), semnal care informează că microprocesorul este pregătit pentru o operație de citire, deci memoria sau dispozitivele de intrare pot depune date pe magistrală;

- $\overline{\text{WR}}$ (Write), semnal activ pe nivel coborât, care permite înscrierea datei prezente pe magistrală în memorie sau în dispozitivul de ieșire.

Simultan cu activarea semnalelor menționate mai sus, microprocesorul orientează magistrala de date spre intrare, respectiv spre ieșire.

Blocul de secvențiere și comandă controlează și regimurile de funcționare ale microprocesorului, după cum urmează:

a) regimul de așteptare (WAIT), prin intermediul a două linii:

- READY - atunci când este activ (READY=1), înseamnă că data cerută din exterior este disponibilă pe magistrala de date; în caz contrar (READY=0), microprocesorul va trece în regimul de așteptare;

- WAIT - semnal generat de microprocesor atunci când READY este găsit pe nivel "0" logic, prin care se specifică faptul că microprocesorul se află în regimul de așteptare; din acest regim se iese după ce memoria sau dispozitivul I/E adresat a depus data pe magistrala de date și semnalul READY a revenit pe nivel "1" logic.

b) regimul de cedare de magistrale, prin blocarea funcționării microprocesorului (HOLD), este de asemenea controlat prin două linii:

- HOLD - semnal de intrare, prin care un alt dispozitiv solicită accesul la resursele sistemului prin intermediul magistralelor de adrese, de date și de comandă, aflate de obicei sub controlul microprocesorului;

- HLDA (HoLD Acknowledge) - semnal emis de microprocesor, care confirmă dispozitivului solicitant acceptarea cererii de cedare de magistrale, după ce în prealabil microprocesorul și-a trecut liniile de adrese și de date în starea de înaltă impedanță.

Intrarea în acest regim este memorată de către microprocesor prin setarea unui bistabil intern (HOLD flip-flop). Revenirea la funcționarea normală se face prin anularea cererii de blocare (HOLD=0), ocazie cu care se resetează și bistabilul aferent.

c) regimul de întreruperi este acceptat pe o singură linie, mascabilă prin program:

- INT (INTerrupt request) - intrare comandată de dispozitivul I/E care solicită intervenția microprocesorului;

- INTE (INTerrupt Enable) - ieșire a unui bistabil intern, care memorează activarea/dezactivarea prin program a întreruperilor (instrucțiunile cu mnemonicele EI și DI în tab.4.6). Dacă întreruperile sunt activate (INTE=1), la apariția unei cereri pe linia INT se setează un alt bistabil intern (INT flip-flop) și microprocesorul trece la execuția unui program de tratare a întreruperii. În cazul în care întreruperile sunt mascate (INTE=0), o solicitare pe linia INT nu este acceptată.

Inițializarea automatului complex care este microprocesorul se face prin intermediul semnalului RESET. La activarea acestui semnal se realizează, printre altele, următoarele acțiuni:

- se aduce la zero contorul de program (PC = 0000h);
- se dezactivează mecanismul de întreruperi (INTE=0) și se resetează CBB INT;
- se resetează bistabilul HOLD și, ca urmare, HLDA devine "0".

În afara regimurilor arătate mai sus, microprocesorul 8080 poate trece într-un **regim de oprire** (HALT), prin execuția instrucțiunii cu codul 76h și mnemonica HLT. În acest regim, de blocare software, microprocesorul își trece magistralele de date și de adrese în starea de înaltă impedanță. Poate reveni la funcționarea normală prin activarea semnalului RESET, care resetează și un bistabil intern, HLTA (HaLT Acknowledge flip-flop), sau printr-o solicitare de întrerupere.

dacă mecanismul de întreruperi a fost în prealabil validat. Se poate realiza astfel o sincronizare a execuției programului cu evenimente externe, a căror apariție este semnalizată pe linia INT.

4.1.2. Secvențierea operațiilor interne

O *stare mașină* este definită, la acest microprocesor, de durata unui perioade de tact. Pentru efectuarea unui schimb de informații cu exteriorul, în cadrul unui *ciclu mașină*, microprocesorul 8080 are nevoie de maximum 5 stări mașină. Între acestea se pot intercala și stări de așteptare, T_w :

$$M = T_1 + T_2 + T_w + T_3 + T_4 + T_5, \text{ unde } T_w, T_4 \text{ și } T_5 \text{ sunt opționale.}$$

Stările T_1 , T_2 și T_3 se regăsesc în orice tip de ciclu mașină, iar stările T_4 și T_5 depind de specificul operațiilor executate în cadrul ciclului mașină.

Execuția unei instrucțiuni necesită cel puțin un ciclu mașină cu 4 stări (T_1 , T_2 , T_3 și T_4) și maximum 5 cicluri. Drept urmare, pentru un *ciclu instrucțiune* se poate scrie următoarea relație:

$$CI = \sum_{i=1}^5 M_i = \sum_{i=1}^5 \sum_{j=1}^k T_{ij},$$

unde $k=3+5$, în funcție de tipul ciclului M_i . Cea mai complexă instrucțiune la acest microprocesor se realizează în 18 stări mașină. Ținând cont de durata unei stări $T = t_{CY} = 0,48 \div 2 \mu s$, rezultă că acest microprocesor poate executa instrucțiunile între cca. 2 și 9 μs , dacă funcționează cu frecvența maximă de tact.

Secvențierea internă a operațiilor la microprocesorul 8080A este prezentată prin intermediul unei diagrame de tranziții (fig.4.3) și a cronogramei generale a unui ciclu mașină (fig.4.4).

În aceste diagrame se observă că, pe durata stării T_1 , microprocesorul depune pe magistrala de date octetul de stare, simultan cu activarea semnalului SYNC. Apoi, în starea T_2 , sunt testate succesiv liniile READY și HOLD. Dacă în ciclul mașină precedent a fost executată o instrucțiune HLT, tot în T_2 se testează dacă bistabilul de confirmare a intrării în regimul HALT (CBB HLTA) a fost setat. În condițiile satisfacerii uneia din condițiile relației HLTA+READY, microprocesorul introduce, după T_2 , stări de așteptare de tip T_w - dacă numai READY=0 și T_{wHALT} - dacă HLTA=1, indiferent de valoarea lui READY. Rezultă că, în starea T_2 , prioritate va avea regimul de stopare soft (HALT). În fig.4.3 a fost specificată finalizarea nebanală a acestui regim, prin trecerea la regimul de întreruperi.

În fig.4.4 este evidențiat modul în care este luat în considerație semnalul READY, fapt important la proiectarea tehnicii de așteptare pentru sincronizarea cu dispozitive externe lente. Datele de catalog [43] specifică necesitatea unui interval de timp de prestabilire, $t_{SETUP} \geq 120 ns$ (specificat hașurat în fig.4.4), necesar pentru ca semnalul pe linia READY să fie luat în considerație.

Într-un mod similar este testat și semnalul HOLD. Dacă este activ, microprocesorul setează bistabilul intern aferent, după care trece în starea T_3 .

Informațiile de pe magistralele de adrese și date sunt valide până în cea de-a doua jumătate a stării T_3 , ceea ce permite microprocesorului să aducă în registrele interne informația pe care o citește în prima parte a unui ciclu de intrare (v.fig.4.4). După realizarea acestor operații, microprocesorul activează semnalul HLDA și trece liniile $A_{15} \div A_0$ și $D_7 \div D_0$ în starea de înaltă impedanță. Liniile DBIN și \overline{WR} devin inactive, dar nu trec în starea de înaltă impedanță.

În cazul în care ciclul mașină respectiv impune, în T_4 și T_5 se execută operații interne. În regimul HOLD microprocesorul testează dacă CBB HOLD este setat și, în caz afirmativ, își blochează activitatea până când linia HOLD se dezactivează. Acest regim poate fi acceptat și dintr-

o stare T_{WH} dacă întreruperile nu sunt active. Trebuie remarcat faptul că din starea T_{WH} regimurile HOLD și INTR se exclud reciproc, fapt vizibil în fig.4.3.

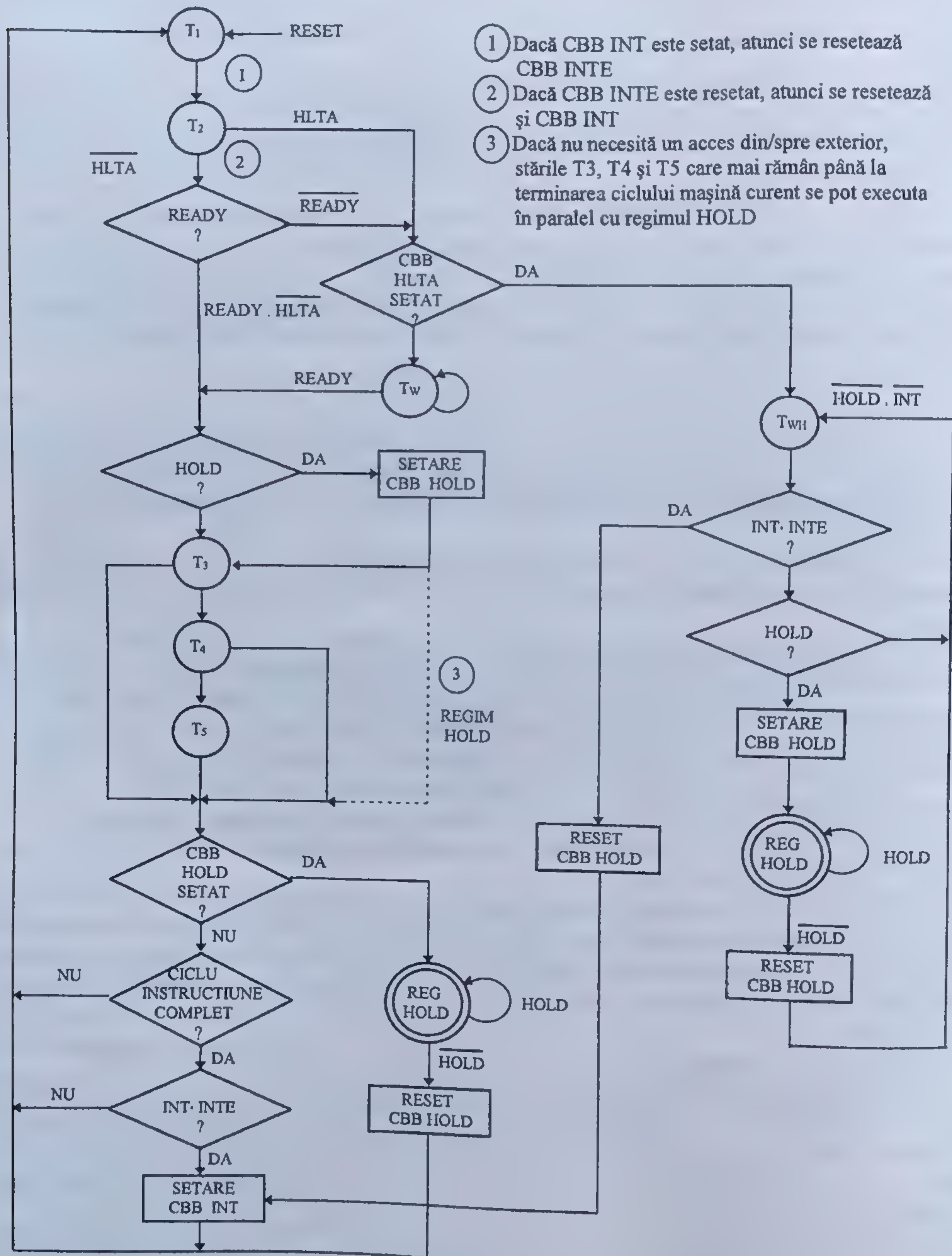


Fig.4.3. Diagrama de tranziții a microprocesorului 8080A

După ieșirea din regimul HOLD, microprocesorul își continuă imediat activitatea cu următorul ciclu mașină, după o prealabilă ștergere a CBB HOLD și preluarea controlului asupra magistralelor de adrese și de date. În cazul în care $READY=1$ și $HOLD=0$, secvențierea operațiilor în ciclurile de scriere sau citire decurge conform fig.4.4, cu starea T_3 și, opțional, T_4 și T_5 .

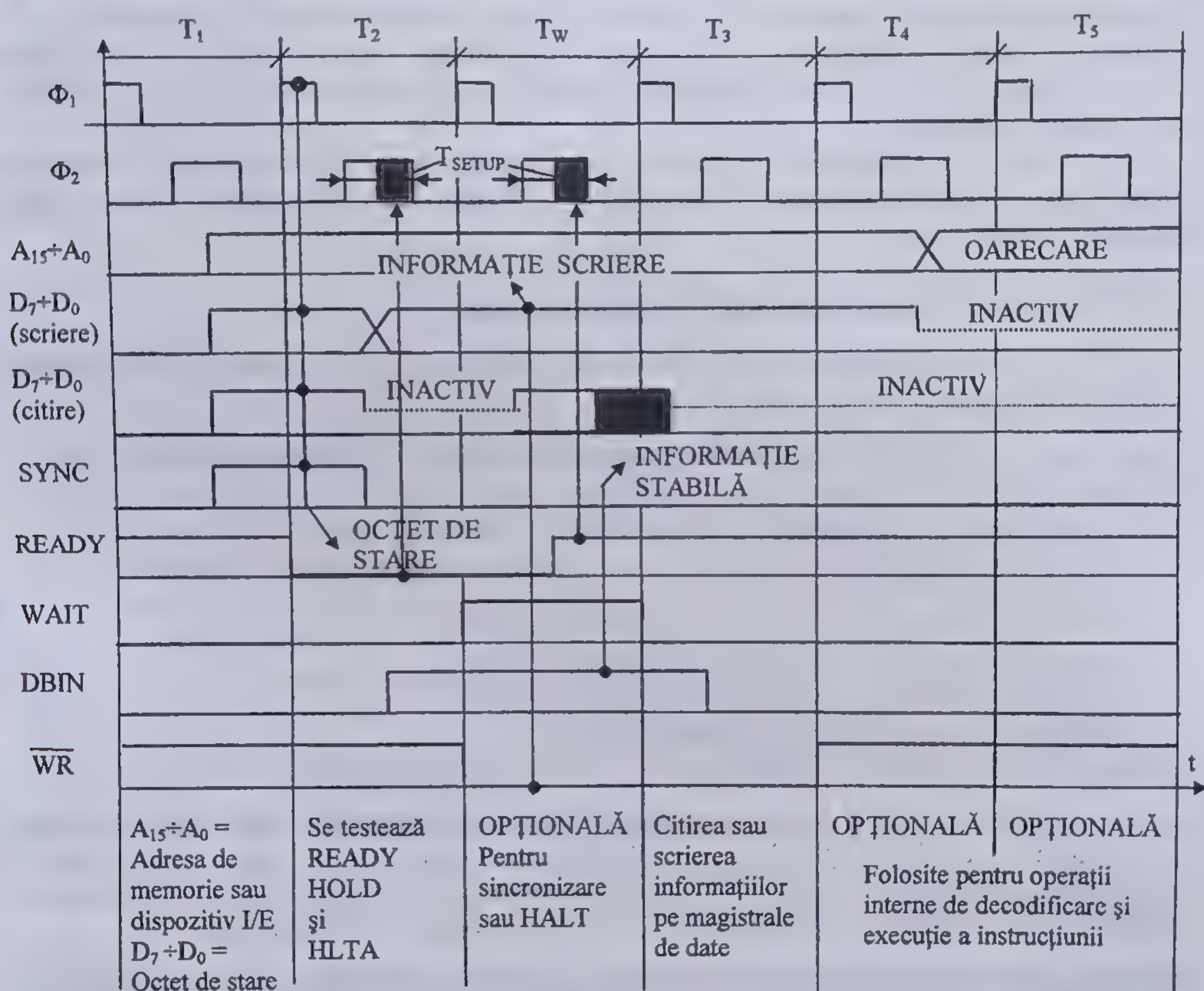


Fig.4.4. Cronograma generală a unui ciclu mașină la microprocesorul 8080A

Secvența T_1-T_3 (eventual și T_4 , T_5) se repetă până la terminarea ciclurilor mașină aferente execuției unei instrucțiuni. Apoi, microprocesorul testează linia INT și, dacă mecanismul de întrerupere a fost în prealabil activat (CBB INTE setat), se setează bistabilul INT și se trece în regimul de întrerupere.

Acest regim începe cu un ciclu M_1 special, numit *ciclu de întrerupere*, în care au loc următoarele acțiuni:

- se dezactivează automat mecanismul de întrerupere, prin resetarea CBB INTE;
- se setează, în octetul de stare, un bit de confirmare a acceptării întreruperii (bitul INTA din tab.4.1);
- se resetează, în octetul de stare, bitul care corespunde comenzii de citire din memorie (bitul MEMR din tab.4.1), specific unui ciclu M_1 normal;
- se inhibă incrementarea conținutului registrului PC, care astfel rămâne încărcat cu adresa instrucțiunii următoare.

Toate acțiunile menționate se efectuează în stările T_1 și T_2 ale ciclului de întrerupere. Deoarece comanda de citire din memorie este inhibată, la detectarea bitului $INTA=1$ din octetul de

stare dispozitivul care solicită întreruperea trebuie să depună pe magistrala de date o informație care să indice microprocesorului adresa subrutinei de tratare a întreruperii. Această informație este adusă în microprocesor în starea T_3 , prin activarea semnalului DBIN (v. fig.4.4) și poate fi codul unei instrucțiuni RST (ReSTart) sau CALL. Instrucțiunile menționate asigură salvarea în stivă a conținutului contorului de program și încărcarea acestuia cu adresa de debut a rutinei de servire a întreruperii (v.tab.4.3). Drept urmare, microprocesorul începe execuția rutinei, iar la terminare trebuie să revină în punctul în care programul a fost întrerupt, prin refacerea din stivă a conținutului numărătorului de adresă.

Din cele prezentate rezultă că proiectantul de sistem trebuie să asigure, pe durata ciclului de întrerupere, condițiile necesare pentru stabilirea dialogului dintre microprocesor și dispozitivul I/E care solicită întreruperea.

4.1.3. Tipuri de cicluri mașină. Octetul de stare

În execuția instrucțiunilor din setul cu care a fost dotat, microprocesorul 8080 utilizează un număr de 10 tipuri de ciclu mașină:

- 1) extragere a codului instrucțiunii (Instruction Fetch Cycle) - este primul ciclu mașină (M_1);
- 2) citire din memorie (Memory Read Cycle);
- 3) scriere în memorie (Memory Write Cycle);
- 4) citire din stivă (Stack Read Cycle);
- 5) scriere în stivă (Stack Write Cycle);
- 6) intrare (Input Read Cycle), de la un dispozitiv de intrare;
- 7) ieșire (Output Write Cycle), către un dispozitiv de ieșire;
- 8) acceptare întrerupere (Interrupt Acknowledge Cycle);
- 9) confirmare oprire (Halt Acknowledge Cycle);
- 10) acceptare întrerupere din starea de oprire (Interrupt Acknowledge While Halt).

Identificarea tipului de ciclu mașină se realizează prin octetul de stare, transmis în starea T_1 pe magistrala de date (fig.4.4). Fiecărui bit din acest octet îi corespunde o informație bine definită despre ciclul care urmează a fi executat, în conformitate cu tabelul 4.1. În acest tabel sunt indicate și valorile pe care le iau biții de stare în funcție de tipul de ciclu mașină.

Tab.4.1.

Tab.4.1

Informația de stare - furnizată de 8080		Tipul de ciclu mașină										
		1	2	3	4	5	6	7	8	8'	9	10
D ₀	INTA	0	0	0	0	0	0	0	1	0	0	1
D ₁	WO	1	1	0	1	0	1	0	1	1	1	1
D ₂	STACK	0	0	0	1	1	0	0	0	0	0	0
D ₃	HLTA	0	0	0	0	0	0	0	0	0	1	1
D ₄	OUT	0	0	0	0	0	0	1	0	0	0	0
D ₅	MI	1	0	0	0	0	0	0	1	0	0	1
D ₆	INP	0	0	0	0	0	1	0	0	0	0	0
D ₇	MEMR	1	1	0	1	0	0	0	0	0	1	0
Semnale de comandă - furnizate direct la pini de către 8080												
17	DBIN	1	1	0	1	0	1	0	1	1	0	1
18	WR	1	1	0	1	0	1	0	1	1	1	1

Tab. 4.1 (cont.)

Magistrala de comandă - generată de 8228												
24	MEMR	0	0	1	0	1	1	1	1	1	1	1
26	MEMW	1	1	0	1	0	1	1	1	1	1	1
25	I / OR	1	1	1	1	1	0	1	1	1	1	1
27	I / OW	1	1	1	1	1	1	0	1	1	1	1
23	INTA	1	1	1	1	1	1	1	0	0	1	0

Semnificația fiecărui bit din octetul de stare este importantă atât pentru înțelegerea mecanismului de identificare a tipului de ciclu cât și pentru proiectant, care trebuie să extragă din acest octet semnalele pentru magistrala de comandă a sistemului.

INTA (INTerrupt Acknowledge) - confirmă faptul că o solicitare de întrerupere a fost acceptată.

Acest bit apare numai în ciclul mașină de întrerupere (8) și de întrerupere din HALT (10). În conjuncție cu semnalul DBIN, determină depunerea de către dispozitivul solicitant, pe magistrala de date, a opcodului unei instrucțiuni RST *n* sau CALL *addr*, care determină ca procesorul să înceapă execuția rutinei de servire a întreruperii. Dacă dispozitivul solicitant a utilizat opcodul instrucțiunii CALL *addr*, ciclul de tipul 8 este urmat de alte două cicluri de tipul 8', în care nu mai sunt activați biții M1 și INTA. Aceste două cicluri sunt utilizate de microprocesor pentru citirea celor doi octeți ai adresei, sub controlul circuitului 8228 (v. §4.1.4.2).

WO (Write Output) - evidențiază faptul că ciclul mașină va fi unul de scriere în memorie sau într-un dispozitiv I/E și apare în ciclurile de tipul 3, 5 sau 7. În conjuncție cu semnalul WR, această informație asigură, în starea T₃ (fig.4.4), activarea semnalului de scriere în exterior (memorie sau dispozitiv de ieșire).

STACK - indică faptul că microprocesorul urmează să lucreze cu stiva; drept urmare, pe magistrala de adrese se depune conținutul indicatorului de stivă (SP).

HLTA (HaLT Acknowledge) - este prezent în ciclurile de tipul 8 și 9, semnalizând în exterior că procesorul s-a oprit în urma execuției unei instrucțiuni HLT.

OUT (OUTput) - indică faptul că urmează să se execute o operație de scriere într-un dispozitiv de ieșire și apare numai în ciclurile de tipul 7. Informația de pe magistrala de date se înscrie în portul de ieșire în starea T₃, ca și în cazul bitului WO, cu ajutorul semnalului WR.

M1 - indică primul ciclu mașină al unei instrucțiuni (Machine Cycle One). Ca atare, el apare în ciclurile FETCH (1), dar și în ciclurile de întrerupere (8 sau 10), când informația forțată din exterior pe magistrala de date este interpretată ca opcod al unei instrucțiuni.

INP (INPut) - indică faptul că magistrala de adrese conține adresa unui dispozitiv de intrare, iar informația de pe magistrala de date este citită pe durata activării semnalului DBIN.

MEMR (MEMory Read) - specifică faptul că magistrala de date va fi folosită într-un ciclu de citire a memoriei, când informația este strobată cu DBIN. Acest bit este prezent în toate ciclurile mașină care necesită o citire din memorie: extragerea instrucțiunii (1), citire din memorie (2) sau din stivă (4), precum și în ciclul HALT (9).

4.1.4. Unitatea centrală de procesare cu 8080A

Pentru îndeplinirea tuturor atribuțiilor care revin unei UCP, microprocesorul 8080A mai are nevoie de două circuite auxiliare, care să-i furnizeze semnalele de tact și, respectiv, să genereze magistrala de comandă.

4.1.4.1. Circuitul pentru generarea semnalelor de tact și de comandă

Din cele arătate în §4.1.1 rezultă că acest microprocesor necesită un semnal de sincronizare cu două faze, de nivel MOS și cu o evoluție temporală în conformitate cu fig.4.2.

Firma Intel a realizat dispozitivul 18224 care, pe lângă generarea semnalelor de tact Φ_1 și Φ_2 , furnizează și o parte din semnalele de comandă necesare la implementarea unui sistem cu microprocesor 8080. Din acest motiv, 8224 a fost denumit *circuit generator de tact și comandă* (Clock Generator and Driver) și are schema bloc din fig.4.5. Dispozitivul este integrat într-o capsulă cu 16 pini și se alimentează cu două tensiuni, $V_{CC} = 5V$ și $V_{DD} = +12V$, pentru asigurarea nivelului MOS al semnalelor Φ_1 și Φ_2 [42].

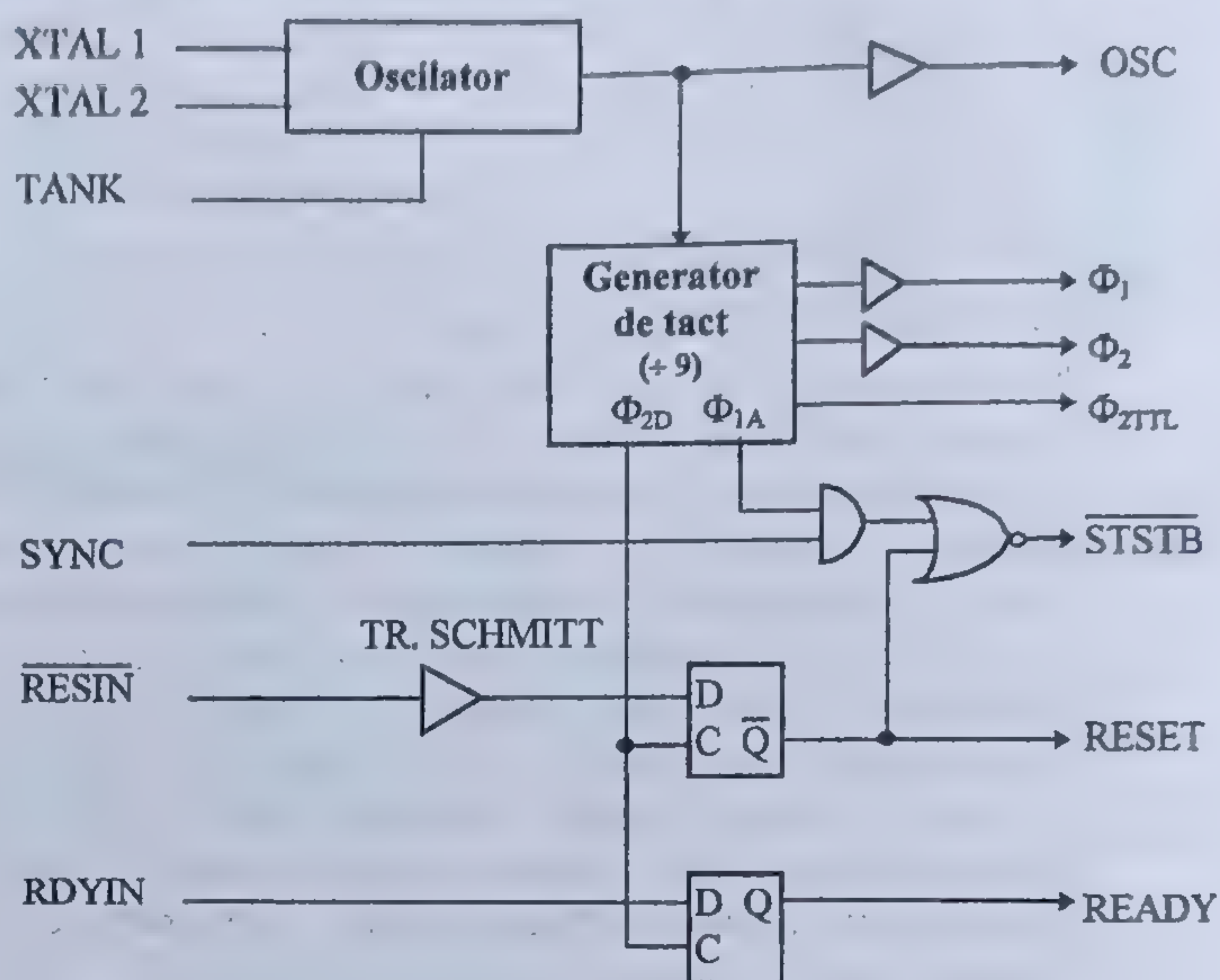


Fig.4.5. Schema bloc a dispozitivului I8224

Circuitul conține un oscilator intern, pilotat cu rezonator de cuarț extern, a cărui frecvență de oscilație poate fi între 4,5 și 18,432 MHz și care se conectează la liniile XTAL1 și XTAL2. Se pot folosi și cristale care lucrează pe armonica a 3-a (cu $f_{max} = 27MHz$) sau rezonatoare serie, care se conectează la terminalul TANK (overtone mode). Ieșirea oscilatorului se aplică unui generator de tact divizor prin 9, precum și în exterior, printr-un amplificator (OSC). În afara semnalelor Φ_1 și Φ_2 , de 12V, acest bloc mai generează și un semnal de nivel TTL, Φ_{2TTL} , util pentru sincronizarea funcționării unor dispozitive externe.

De asemenea, 8224 conține o logică formată din porți și bistabile pentru:

- generarea unui semnal de sincronizare, \overline{STSTB} (STatus STroBe), de cca. 50ns, utilizabil pentru a comanda un registru cu memorare (latch) pentru reținerea informației de stare transmisă de 8080 pe magistrala de date, odată cu semnalul SYNC. În fig.4.6 este prezentată cronograma pentru $t_{CY} = 500ns$, în care se evidențiază generarea semnalului \overline{STSTB} în avans față de frontul ascendent al lui Φ_1 (Φ_{1A}), astfel încât strobarea magistralei de date să se facă atunci când informația de stare este stabilă.

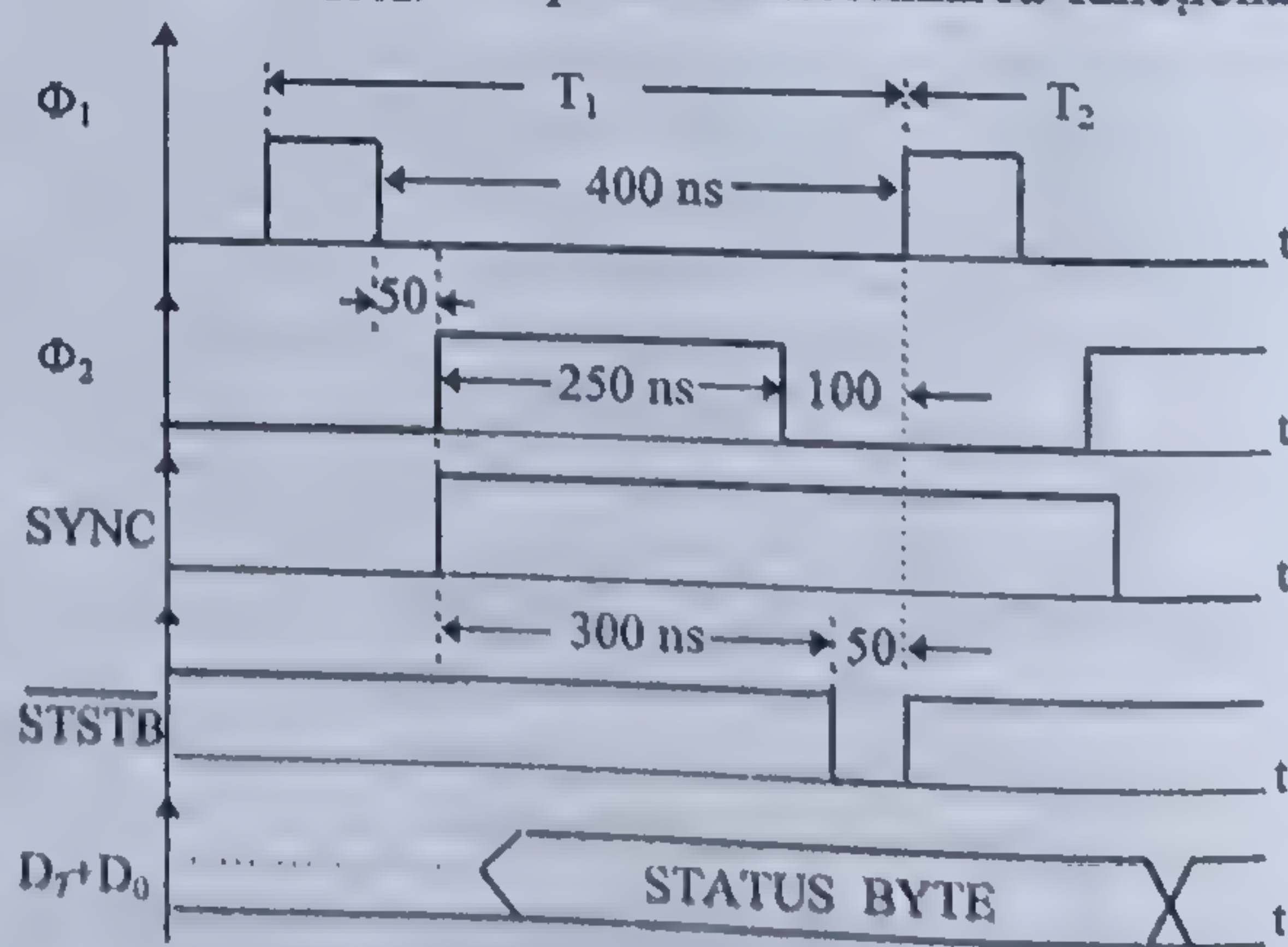


Fig.4.6. Cronograma generării semnalului \overline{STSTB}

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

- generarea semnalelor RESET și READY, sincronizate cu Φ_2 și de nivel logic corespunzător ($V_{Hmin} = 3,65V$). Existența unui trigger Schmitt la intrarea \overline{RESIN} (RESet INput) permite utilizarea unui circuit simplu de resetare, bazat pe o rețea RC (v.fig.1.5). De asemenea, în cazul în care este necesară tehnica de așteptare, la intrarea RDYIN (ReaDY INput) se asigură timpul de prestabilire impus ($t \geq 190$ ns).

4.1.4.2. Circuitul pentru generarea magistralei de comandă

În paragraful precedent s-a văzut că microprocesorul 8080 nu generează direct semnalele pentru controlul dialogului cu memoria și cu dispozitivele I/E. În schimb, acesta furnizează un octet de stare la începutul fiecărui ciclu mașină, care conține informațiile necesare generării semnalelor de comandă. Faptul că această informație este transmisă numai pe durata stărilor T_1 și T_2 , iar semnalele de comandă trebuie să existe și în celelalte stări ale unui ciclu mașină, impune folosirea unor dispozitive externe, care să genereze corespunzător semnalele pe magistrală. De asemenea, întrucât la magistrala de date se conectează toate dispozitivele necesare în sistem (v.fig.1.1), chiar în cazul unui număr redus de circuite MOS (≥ 7) se depășește capacitatea de încărcare a acesteia (fan-out). Ca urmare, este necesară de obicei amplificarea magistralei de date, cu conservarea proprietăților sale: bidirecționalitatea și regimul 3-stări.

Cu cele arătate în §4.1.3 pot fi proiectate structurile logice necesare folosind porți și registre standard. Dar, mai avantajos este să se folosească în acest scop circuitul specializat 18228, realizat de Intel. Deoarece îmbină în același dispozitiv integrat ambele funcții arătate mai sus, a fost denumit *controler de sistem și driver de magistrală* (System Controller and Bus Driver). Este o structură mediu integrată, realizată sub forma unei capsule cu 28 de pini și alimentată la +5V.

Schema bloc internă este prezentată în fig.4.7 și conține două părți distincte funcțional:

- un driver bidirecțional de magistrală, care asigură pe liniile $DB_7 \div DB_0$ un fan-out de 6 sarcini TTL ($I_{OL} = 10mA$), în timp ce liniile $D_7 \div D_0$ consumă $250\mu A$;

- o logică de generare a semnalelor de comandă, prin reținerea informației de stare într-un tampon strobos cu

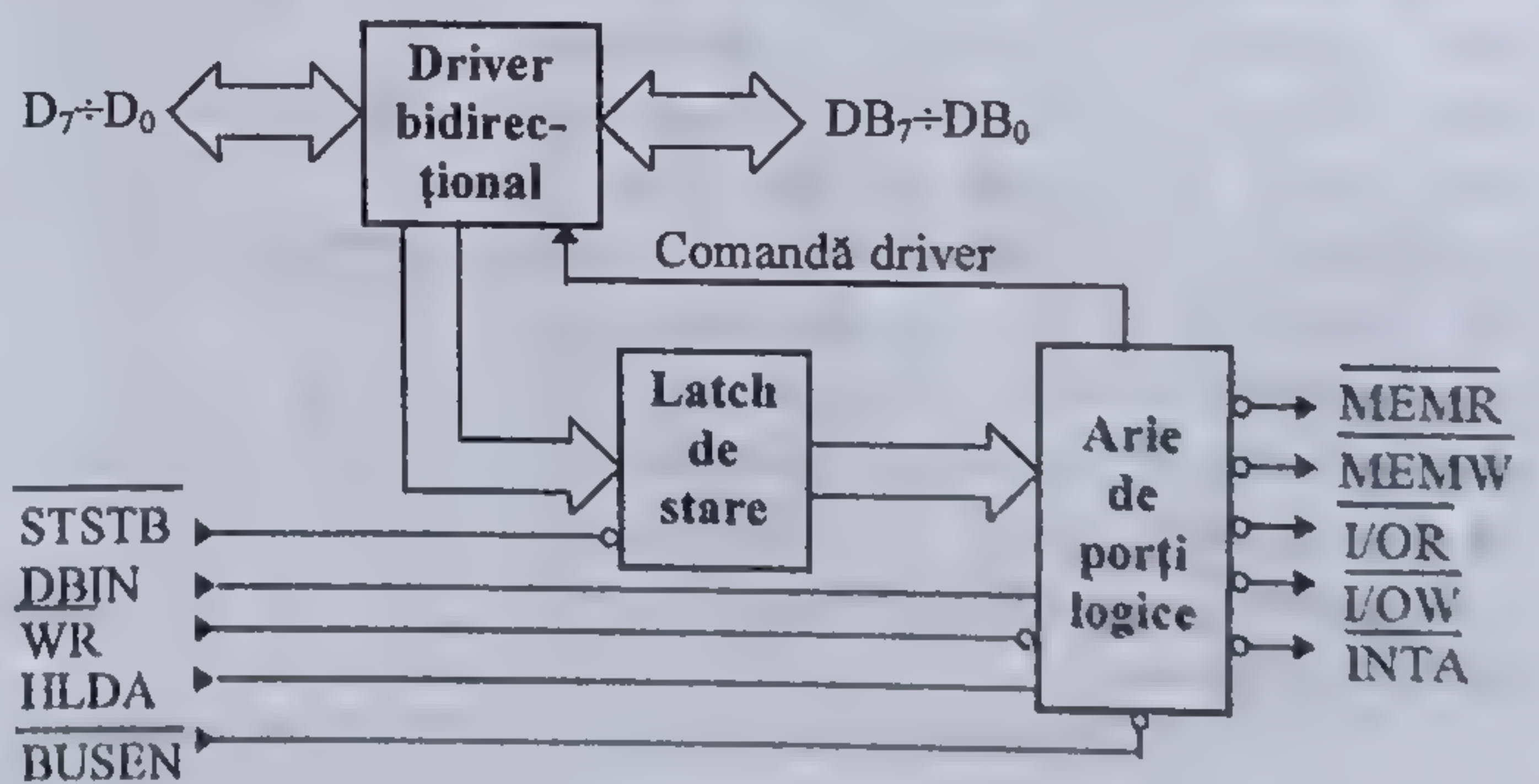


Fig.4.7. Schema bloc a dispozitivului 18228

semnalul \overline{STSTB} și mixarea cu semnalele \overline{DBIN} și \overline{WR} într-o arie de porți logice. Se obțin astfel semnalele necesare magistralei de comandă a sistemului, care respectă următoarele relații:

$$\overline{MEMR} = \overline{D_7} \cdot \overline{DBIN} - \text{semnal de citire din memorie};$$

$$\overline{MEMW} = \overline{D_4} \cdot \overline{WR} - \text{semnal de scriere în memorie};$$

$$\overline{I/OR} = \overline{D_6} \cdot \overline{DBIN} - \text{semnal de citire dintr-un port de intrare};$$

$$\overline{I/OW} = \overline{D_4} \cdot \overline{WR} - \text{semnal de scriere într-un port de ieșire};$$

$$\overline{INTA} = (\overline{D_0} + \overline{D_1} \cdot \overline{D_6} \cdot \overline{D_7}) \cdot \overline{DBIN} - \text{semnal de răspuns la o solicitare de întrerupere}.$$

Valorile logice ale acestor semnale sunt prezentate în ultima parte a tab.4.1 și depind, așa cum o arată și relațiile de mai sus, atât de biții D_0 , D_1 , D_4 , D_6 și D_7 din octetul de stare, cât și de

cele două semnale de comandă furnizate direct la pini de 8080, \overline{DBIN} și \overline{WR} . În ecuația pentru \overline{MEMW} s-a folosit $\overline{D_4}$ pentru a se specifica faptul că microprocesorul nu va scrie într-un port, ci în memorie, ceea ce nu se poate stabili cu informația generală din D_1 (\overline{WO}). În ecuația pentru \overline{INTA} s-a folosit D_0 pentru ciclii 8 și 10 și $D_1 \cdot \overline{D_6} \cdot \overline{D_7}$, pentru a surprinde activarea semnalului și în ciclurile de întrerupere suplimentare, de tipul 8', în care nu mai este activat bitul D_0 . Astfel este posibilă generarea impulsurilor \overline{INTA} corespunzătoare, pentru preluarea completă a unei instrucțiuni CALL de 3 octeți, într-un ciclu de întrerupere.

Primele patru linii sunt de tip TS (Tri-State), iar linia \overline{INTA} este de tip cu colectorul în gol. De asemenea, liniile $DB_7 \div DB_0$ sunt de tip TS, pentru a fi trecute în starea de înaltă impedanță odată cu magistrala microprocesorului, în regimul de HOLD (deci atunci când semnalul HLDA devine activ). Liniile de comandă pot fi trecute în starea de înaltă impedanță, prin dezactivarea liniei \overline{BUSEN} (BUS ENable) de către noul coordonator, în momentul în care acesta preia controlul sistemului.

Trebuie menționat faptul că dacă dispozitivul 8228 are linia de ieșire \overline{INTA} conectată la +12V printr-un rezistor [24, 43], atunci acesta forțează depunerea pe magistrala de date a opcodului unei instrucțiuni de tip RESTART (RST 7, cu opcodul FFh) în primul ciclu de acceptare a întreruperii (când $D_0=1$).

Din cele prezentate rezultă că dispozitivul 8228 generează numai semnalele strict necesare pentru control, restul informațiilor din octetul de stare fiind ignorate. De multe ori este necesar să se cunoască toate informațiile de stare, mai ales în cazul testării corecte funcționării a microprocesorului. Folosind un dispozitiv 8212, cu funcționare în regimurile DATA IN - DATA LATCH, se poate obține accesul la întregul octet de stare. În fig.4.8 este prezentată schema de conectare la microprocesor a unui astfel de dispozitiv.

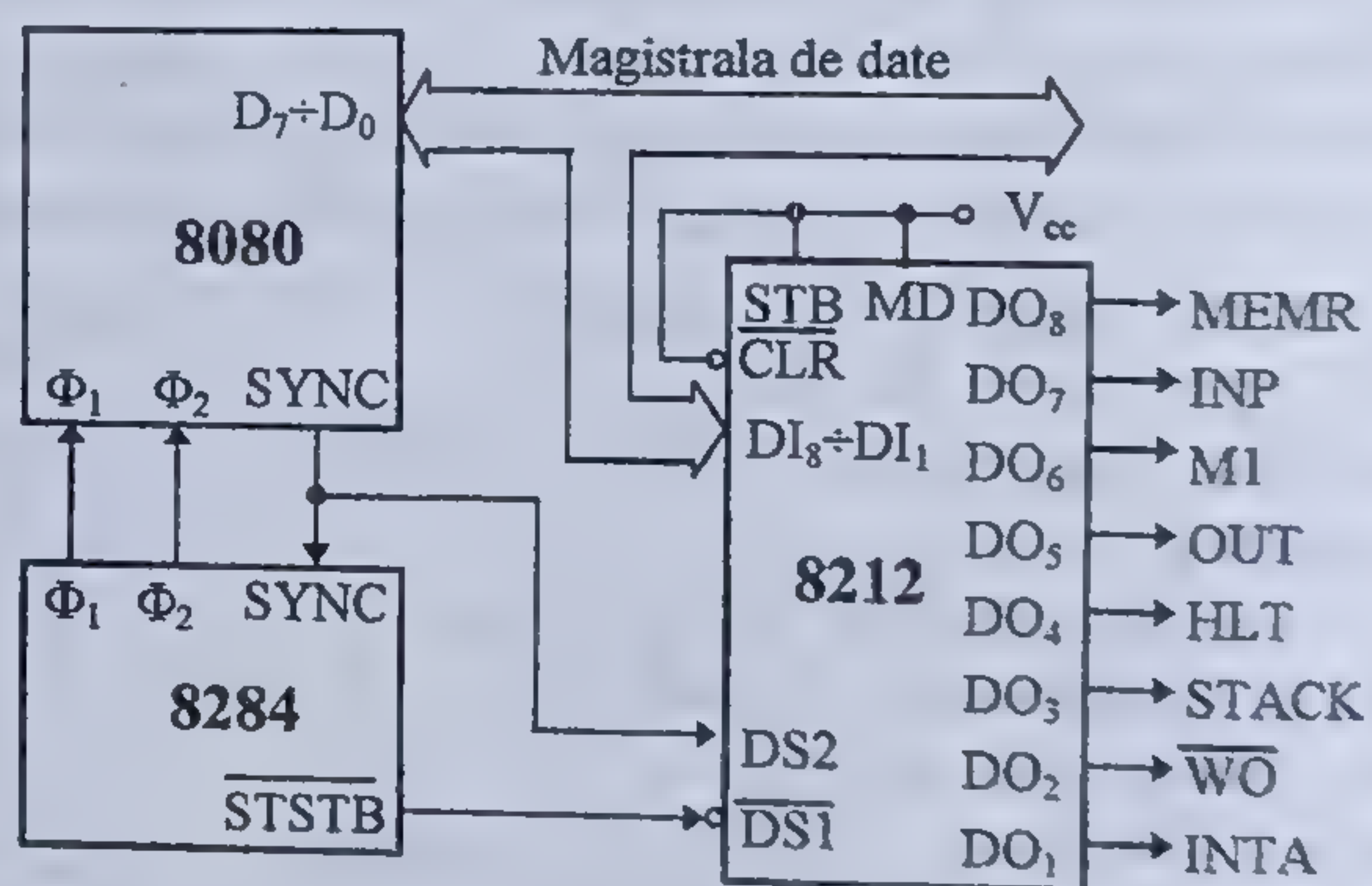


Fig.4.8. Reținerea octetului de stare cu circuitul 8212

În conformitate cu tabelul din fig.1.25b este necesar ca $STB=MD=1$, iar controlul pentru fiecare ciclu mașină se face prin liniile $\overline{DS1}=\overline{STSTB}$ (de la 8224) și $DS2=SYNC$. La începutul fiecărui ciclu mașină, în starea T_1 , se generează semnalele SYNC și \overline{STSTB} , care asigură satisfacerea relației $\overline{DS1} \cdot DS2=1$, deci activarea regimului DATA IN. Intrările $DI_8 \div DI_1$ ale portului 8212 fiind conectate la magistrala de date a microprocesorului, acesta memorează octetul de stare odată cu activarea semnalului \overline{STSTB} . La dezactivarea acestuia, $\overline{DS1} \cdot DS2=0$, fapt ce conduce la trecerea lui 8212 în regim DATA LATCH și reținerea octetului de stare până la un nou ciclu mașină.

4.1.4.3. Structura unității centrale și a unui sistem cu 8080A

Întrucât circuitele 8224 și 8228 sunt direct compatibile cu microprocesorul 8080A, realizarea UCP implică numai interconectarea liniilor cu același nume. De asemenea, trebuie prevăzut un circuit de inițializare a UCP și, dacă este cazul, circuite de amplificare a magistralei de adrese (spre exemplu, 2×8212 în regim DATA IN). În fig.4.9 este prezentată schema generală a unui sistem cu 8080A, în care este evidențiată unitatea centrală de procesare.

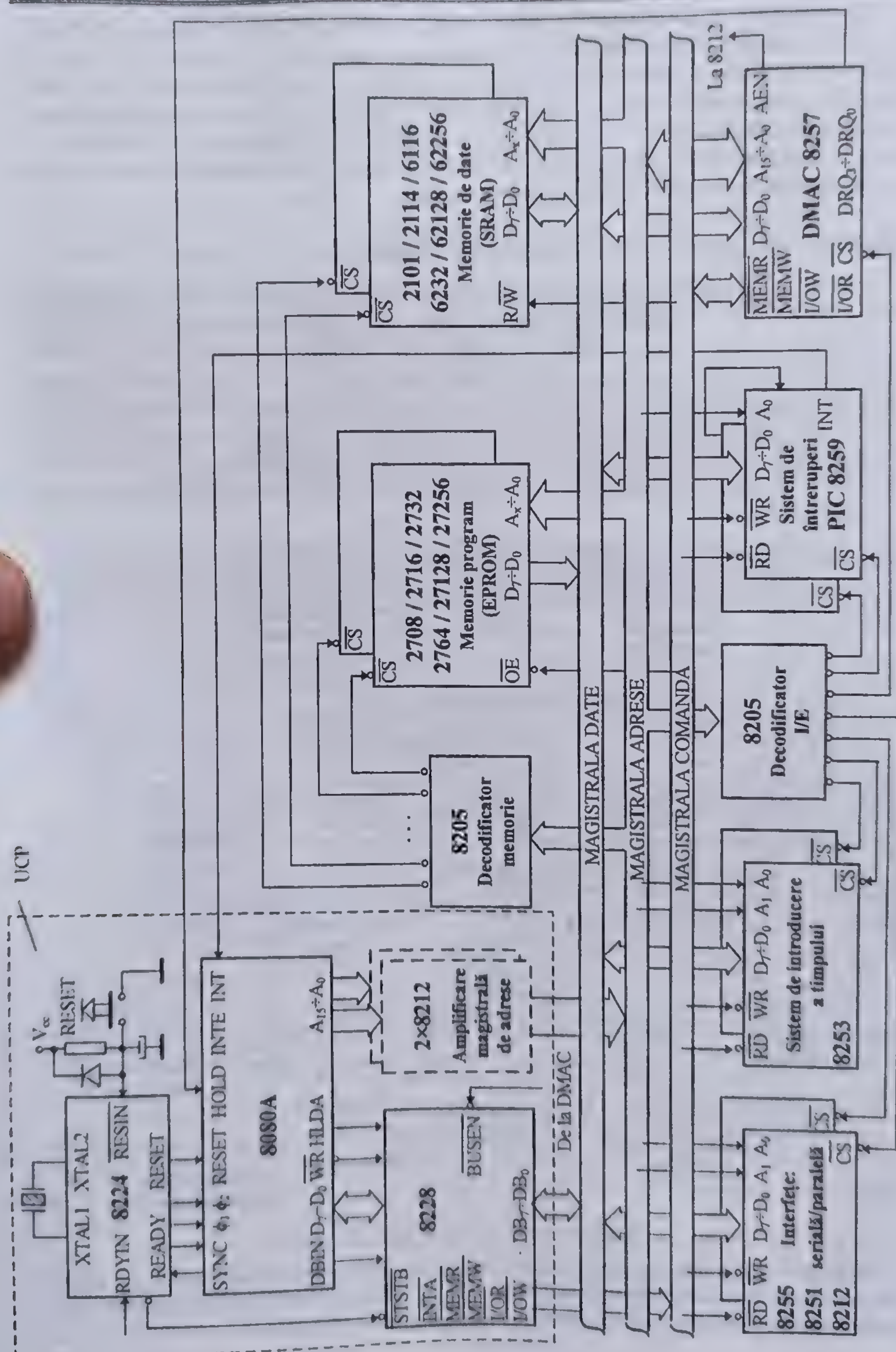


Fig.4.9. Structura unui microsistem cu 8080A

S-a considerat cazul selecției cu decodificare a circuitelor de memorie și a porturilor. Subsistemul de memorie se determină în conformitate cu cele prezentate în cap.1. Porturile I/E specificate sunt compatibile cu microprocesorul și se conectează direct la magistralele sistemului; fiind dispozitive programabile, acestea trebuie să fie corespunzător inițializate și configurate prin program de către microprocesor. La realizarea sistemului de întreruperi și a sistemului de introducere a timpului, în funcție de particularitățile acestora, este necesară o interconectare specifică, așa cum se va arăta în capitolele următoare.

4.2. Sisteme cu microprocesorul Intel 8085A

Așa cum s-a menționat anterior, microprocesorul 8085A reprezintă varianta îmbunătățită a lui 8080A. Produs în 1976, în tehnologie NMOS, acesta este ultimul microprocesor de 8 biți, din seria de uz general, al firmei Intel. Din punct de vedere HW, 8085 înglobează într-un singur circuit integrat atât funcțiile lui 8080A cât și pe cele ale circuitelor 8224 și 8228. Semnalele de comandă nu mai sunt multiplexate, în schimb octetul inferior al adresei de 16 biți este multiplexat în timp, pe magistrala de date. Din punct de vedere SW, 8085A este 100% compatibil cu predecesorul său, orice program scris pentru 8080A putând fi executat la nivel de cod mașină și pe 8085A. Setul de instrucțiuni conține doar două instrucțiuni suplimentare, pentru controlul noilor facilități cu care a fost prevăzut 8085A.

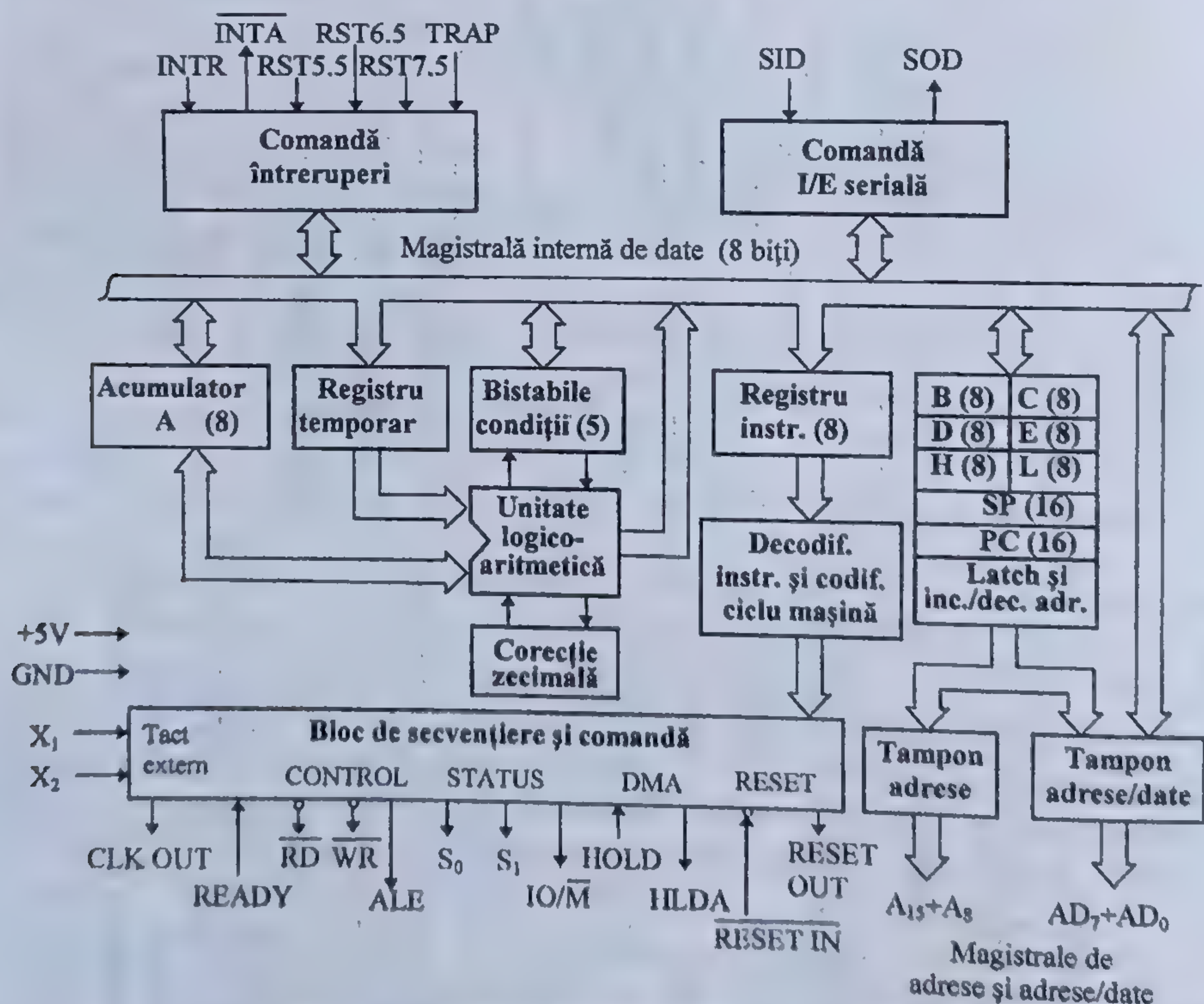


Fig.4.10. Schema bloc a microprocesorului 8085A

În fig.4.10 este prezentată schema bloc a microprocesorului 8085A, în care se pot vedea o parte din asemănările și deosebirile în raport cu 8080A. Astfel, principalele îmbunătățiri aduse acestui microprocesor în comparație cu predecesorul său sunt următoarele:

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

- necesită o singură tensiune de alimentare, de +5V;
- frecvența maximă de lucru este de 3MHz, având oscilatorul inclus pe cip. În exterior se conectează un cristal de cuarț pe intrările X_1 și X_2 , sau un generator extern de tact pe intrarea X_1 . Durata unui ciclu de tact (t_{CY}) poate fi la 8085A cuprinsă între 0,32 și 2μs.

- posibilitatea de tratare multiplă a întreruperilor prin intermediul a 5 linii dedicate: INTR, RST 5.5, RST 6.5, RST 7.5 și TRAP. Intrarea INTR este funcțional identică cu INT de la 8080A și are asociată o ieșire de confirmare a acceptării întreruperii, \overline{INTA} . Ca și INTR, liniile RST sunt linii de întrerupere mascabile (pot fi validate sau inhibitate prin program). În schimb, ele nu necesită o confirmare externă a acceptării solicitărilor, întrucât microprocesorul 8085 execută automat instrucțiuni de tip RESTART (v. §4.3) implementate în hardware. Codurile 5.5, 6.5 și 7.5 specifică indirect adresa de salt corespunzătoare fiecărei linii de întrerupere, adresă care se obține prin multiplicarea cu 8 a acestor coduri (de ex. $5,5 \times 8 = 44 = 002Ch$).

Linia RST 7.5 are prioritatea cea mai mare, iar RST 5.5 - prioritatea cea mai mică, dar superioară celei a liniei INTR. În fine, linia TRAP are prioritate maximă și asigură execuția, implementată prin HW, a unei instrucțiuni RST 4.5, indiferent dacă sistemul de întreruperi al microprocesorului este sau nu activ; aceasta este o linie de întrerupere *nemascabilă*.

În tabelul 4.2 se prezintă adresele de salt, prioritățile, precum și modul de activare a celor cinci linii de întrerupere.

Tab.4.2.

Linia de întrerupere	Adresa de salt	Nivelul de prioritate	Modul de activare	Tipul întreruperii
TRAP	0024h	1 (max)	front și nivel	nemascabilă
RST 7.5	003Ch	2	front pozitiv	mascabilă
RST 6.5	0034h	3	nivel logic "1"	mascabilă
RST 5.5	002Ch	4	nivel logic "1"	mascabilă
INTR	dată de instrucțiunea RST n sau CALL $addr$	5 (min)	nivel logic "1"	mascabilă

Pentru activare, liniile INTR, RST 5.5 și RST 6.5 trebuie menținute pe nivel logic "1" până când sunt eșantionate de microprocesor: la sfârșitul execuției unei instrucțiuni, în penultimul ciclu de tact. Linia RST 7.5 este activă numai pe front crescător; apariția acestuia este memorată într-un bistabil intern, care este resetat după servire sau după o mascare software.

Linia nemascabilă, TRAP, activă atât pe front pozitiv cât și pe nivel, este achitată numai dacă este menținută pe nivel ridicat până la strobare. În acest mod se evită activarea mecanismului de întrerupere la impulsuri logice false (logic glitches). De obicei, această linie este rezervată pentru situații "catastrofale" în care se poate afla sistemul, cum ar fi iminenta dispariție a tensiunii de alimentare. Pentru astfel de situații, proiectantul de sistem poate folosi activarea liniei TRAP în scopul salvării datelor importante din memoria RAM volatilă (al cărei conținut se pierde la deconectarea tensiunii de alimentare) într-o zonă de memorie RAM nevolatilă (de exemplu alimentată la baterii).

Trebuie specificat și faptul că liniile RST pot fi mascate, independent de linia INTR, printr-o instrucțiune cu mnemonica SIM (Set Interrupt Mask). Resetarea măștii de întrerupere se poate realiza printr-o instrucțiune cu mnemonica RIM (Read Interrupt Mask). Numai aceste două tipuri de instrucțiuni sunt adăugate setului de instrucțiuni al microprocesorului 8080A, ele utilizându-se și la controlul blocului pentru I/E serială.

Din cele arătate mai sus rezultă că 8085A are inclus un controler de gestionare a cererilor multiple de întrerupere, fiind mai potrivit pentru aplicații de timp real decât predecesorul său.

O altă facilități suplimentară importantă a microprocesorului 8085A este posibilitatea transferului serial al datelor prin intermediul a două linii: de intrare - SID (Serial Input Data) și de

ieșire - SOD (Serial Output Data). Linia SID este controlată software prin instrucțiunea RIM; la execuția ei 8085A încarcă bitul de date corespunzător nivelului logic din acel moment al liniei SID în poziția cea mai semnificativă (bitul 7) a acumulatorului. Prin execuția instrucțiunii SIM, bitul cel mai semnificativ al acumulatorului este depus pe linia SOD. Astfel, se poate implementa prin program o comunicație serială simplă, fără alte circuite suplimentare.

În afara îmbunătățirilor menționate mai sus mai pot fi remarcate și altele, care trebuie cunoscute de proiectantul de sistem:

- generarea unui semnal ALE (Address Latch Enable), pentru comanda unui registru (latch) care să rețină de pe magistrala complexă $AD_7 \div AD_0$ octetul inferior al adresei ($A_7 \div A_0$), la începutul fiecărui ciclu mașină;

- controlul vehiculării informației pe magistrale prin intermediul a 3 semnale: \overline{RD} , \overline{WR} și IO/\overline{M} . Ultimul semnal face distincție între dialogul cu dispozitive I/E ("1" logic) și cu memoria ("0" logic);

- starea magistralei de date (Data Bus Status) în principalele cicluri mașină este semnalizată pe liniile S_0 și S_1 , ca în tab.4.3;

- generarea de semnale de resetare (RESET OUT) și de tact (CLOCK OUT - cu frecvența de $1/2$ din cea aplicată pe X_1 , X_2), utile pentru sincronizarea componentelor sistemului cu UCP.

Tab.4.3.

S_1	S_0	Tip ciclu
0	0	HALT
0	1	WRITE
1	0	READ
1	1	FETCH

Celelalte semnale au aceeași semnificație cu omonimele lor

de la 8080A: READY, HOLD, HLDA. Semnalul $\overline{RESET IN}$ are același efect ca și RESET de la 8080A: aducerea la adresa 0000h a registrului PC și resetarea bistabililor INTE (validare întreruperi) și HLDA (confirmare cedare magistrale).

Liniile $A_{15} \div A_8$, $AD_7 \div AD_0$, \overline{RD} și \overline{WR} sunt de tipul TS. Linia IO/\overline{M} trece în starea de înaltă impedanță în regimurile HOLD și HALT.

4.2.1. Secvențierea operațiilor interne la microprocesorul 8085A

Ca și microprocesorul 8080A, 8085A are nevoie, pentru execuția unei instrucțiuni, de 1 până la 5 cicluri mașină. La acest microprocesor un ciclu mașină necesită 3 până la 6 stări (cicluri de tact). Pentru a putea

asigura execuția celor 80 de tipuri de instrucțiuni cu care este dotat, microprocesorul 8085A utilizează 7 tipuri de cicluri mașină. Acestea sunt caracterizate prin semnalele de stare (S_0 , S_1 , IO/\overline{M}) și de comandă (\overline{RD} , \overline{WR} , \overline{INTA}) prezentate în tabelul 4.4.

Tab.4.4.

Tip ciclu	Stare			Comandă		
	S_1	S_0	IO/\overline{M}	\overline{RD}	\overline{WR}	\overline{INTA}
OF (Opcode Fetch)	1	1	0	0	1	1
MR (Memory Read)	1	0	0	0	1	1
MW (Memory Write)	0	1	0	1	0	1
IOR (I/O Read)	1	0	1	0	1	1
IOW (I/O Write)	0	1	1	1	0	1
INA (INT. Ack.)	1	1	1	1	1	0
BI (Bus Idle): DAD	1	0	0	1	1	1
INA	1	1	1	1	1	1
HALT	0	0	HZ	HZ	HZ	1

HZ - stare de impedanță ridicată

În fig.4.11 se prezintă diagrama de tranziții de stare la microprocesorul 8085A. Față de cea de la 8080, apar deosebiri numai în ciclul OF (echivalent M_1), a cărui durată poate fi de 4 sau de 6 cicluri de ceas (CC), în funcție de tipul instrucțiunii.

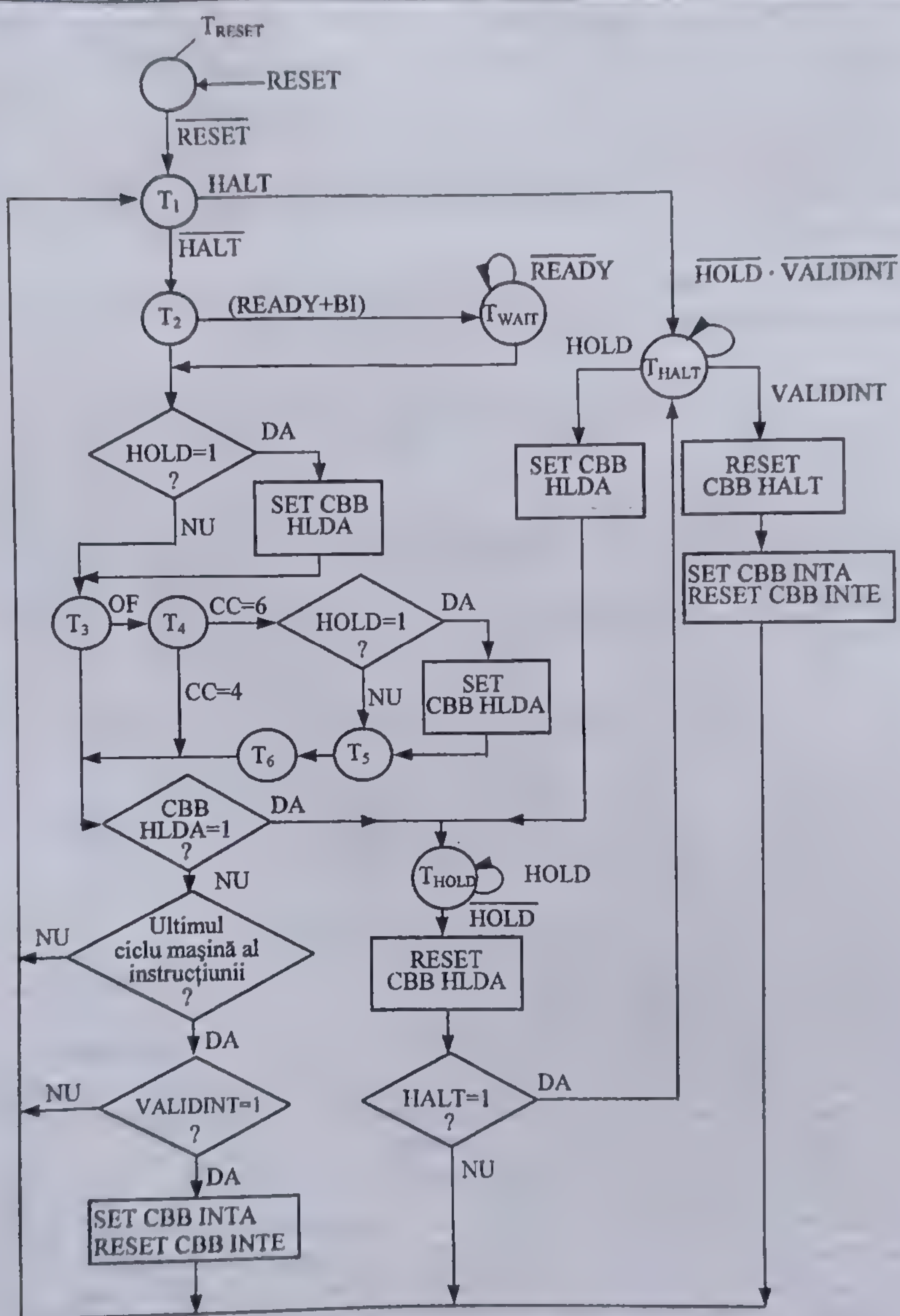


Fig.4.11. Diagrama de tranziție a stărilor la microprocesorul 8085A

Obs.: CC - numărul de cicluri de ceas al ciclului mașină curent;
BI (Bus Idle machine cycle) - ciclu mașină care nu utilizează magistrala sistemului;
VALIDINT (VALid INTerrupt) - există o cerere de întrerupere și întreruperile sunt validate (INT), respectiv nu sunt mascate (RST 5.5, 6.5, 7.5);
CBB HLDA - bistabil intern, de memorare a cererii de cedare de magistrale. Magistralele microprocesorului 8085A sunt trecute în starea de înaltă impedanță după un ciclu de tact de la setarea acestui bistabil.

Ciclul OF este ciclul de citire din memorie a codului operație al instrucțiunii, deci trebuie să fie caracterizat prin $IO/\overline{M}=0$, $S_0=S_1=1$ (FETCH) și activarea semnalului \overline{RD} .

În fig.4.12 este prezentată diagrama de semnale pentru un astfel de ciclu. Se observă execuția operațiilor OUT și IN pe liniile $AD_7 \div AD_0$, necesare depunerii octetului inferior al adresei, urmată de transferul datelor. Și la acest microprocesor în starea T_2 , pe frontul pozitiv al semnalului de tact, se testează linia READY. Dacă $READY = 0$, se inserează stări T_{WAIT} între T_2 și T_3 ; după READY este testată și linia HOLD.

Ciclurile FETCH includ cel puțin încă o stare (T_4) și maximum trei (T_4, T_5, T_6) în care se realizează decodificarea și execuția, în cazul instrucțiunilor de un octet.

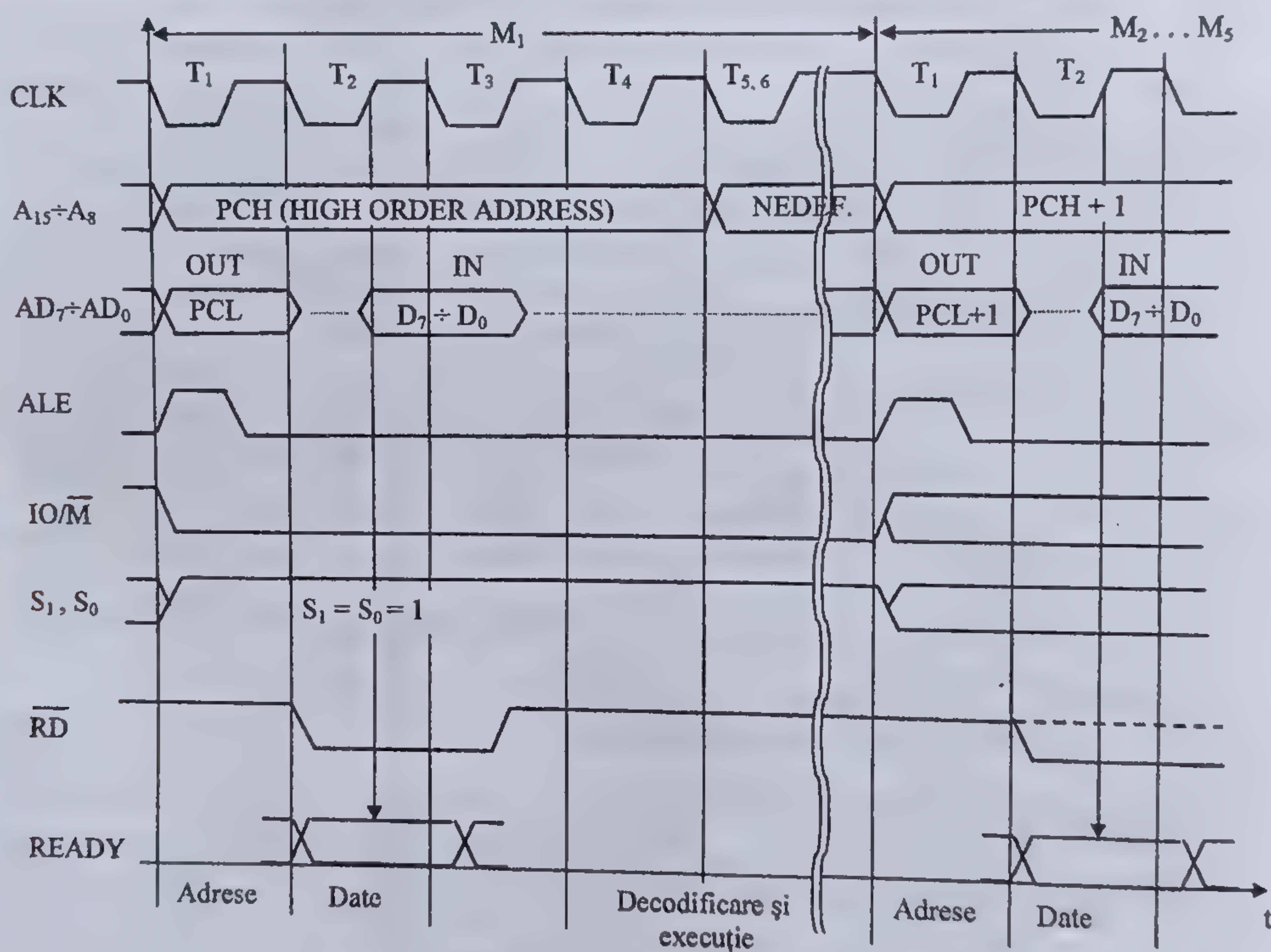


Fig.4.12. Diagrama de semnale pentru un ciclu OF

În ciclurile MR, MW, IOR și IOW se realizează operații de citire sau înscriere asupra memoriei sau dispozitivelor I/E, care durează de obicei trei stări. În fig.4.13 este ilustrată secvențierea operațiilor la execuția unei instrucțiuni cu mnemonica STA *addr*, de 3 octeți (v.tab.4.6 în §4.3.3). Instrucțiunea depune conținutul acumulatorului în memorie, la adresa specificată prin *addr* (de 16 biți). Pentru execuție sunt necesare 4 cicluri mașină: OF - pentru aducerea codului instrucțiunii, MR - pentru aducerea octetului inferior al adresei, low(*addr*) sau, MR - pentru aducerea octetului superior al adresei, high(*addr*), și MW - pentru înscrierea conținutului acumulatorului la adresa specificată pe liniile de adresă $A_{15}+A_0$ (*addr*). Și în aceste cicluri mașină, în T_2 , este testată mai întâi starea liniei READY și apoi cea a liniei HOLD.

Ciclul INA este utilizat pentru confirmarea acceptării unei întreruperi pe linia INTR și se execută în aceeași manieră ca și la microprocesorul 8080A, după ce, în prealabil, în prima stare (T_1) a ciclului special de tratare a întreruperii (M_{INA}), se testează starea liniilor TRAP și RST.

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

Numai în cazul în care cele patru linii de întrerupere de prioritate superioară nu sunt active se continuă ciclul INA. Mai întâi se citește o instrucțiune RST n sau CALL $addr$, forțată din exterior pe magistrala de date de dispozitivul care solicită întreruperea. În tot acest timp, activarea semnalului \overline{RD} este inhibată, aceasta fiind înlocuită de activarea semnalului \overline{INTA} (v.tab.4.4), iar contorul de program este blocat. Apoi, se trece la execuția instrucțiunii primite, care începe cu depunerea în stivă a conținutului registrului PC și continuă cu încărcarea acestuia cu adresa rutinei de tratare a întreruperii.

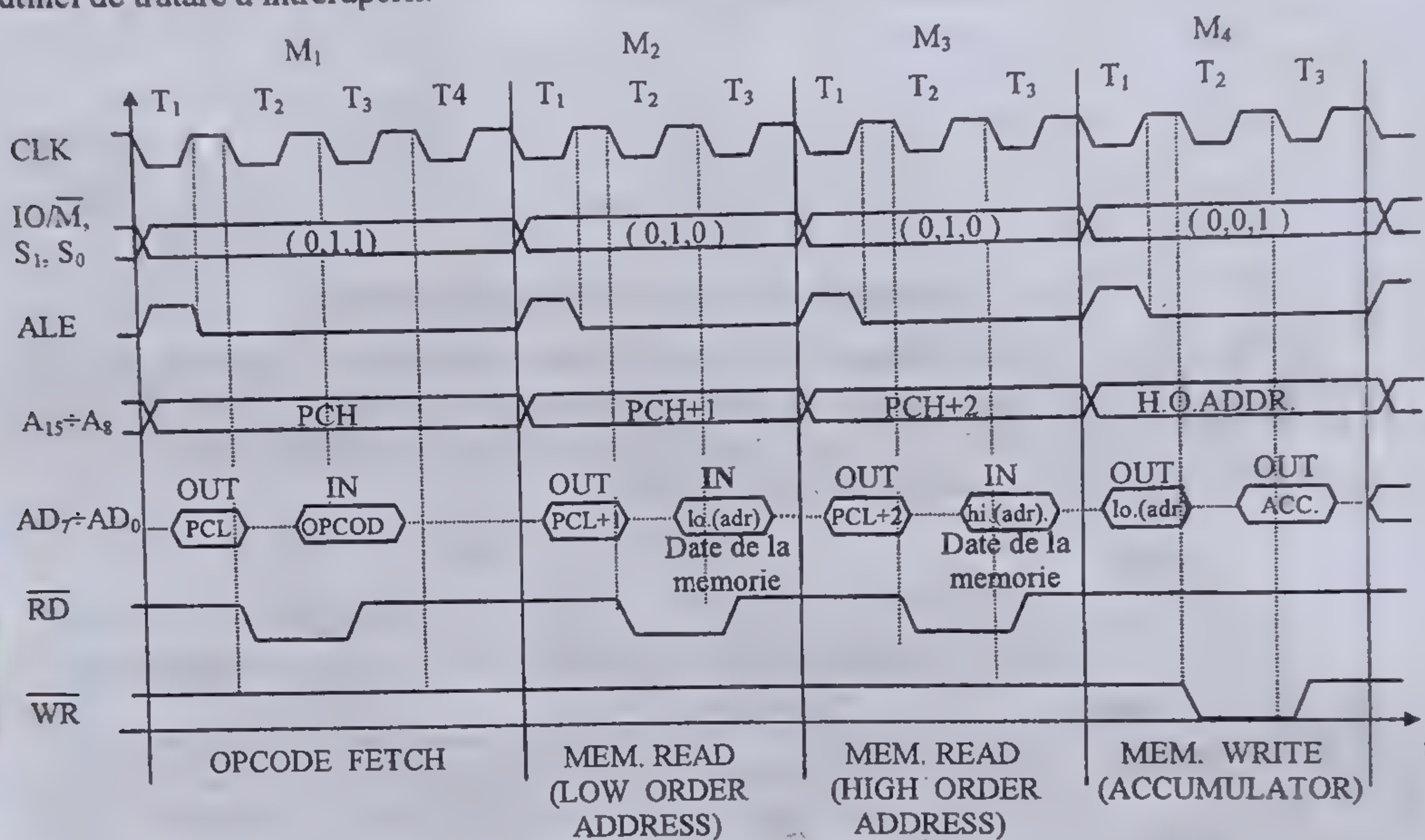


Fig.4.13. Cronograma instrucțiunii STA $addr$

Ciclul BI se referă la trei situații speciale, în care magistrala de date nu este utilizată.

Prima este cea a execuției instrucțiunii cu mnemonica DAD rp (v.tab.4.6 în §4.3.3) când, în ciclurile M_2 și M_3 (după $OF=M_1$), magistrala nu este folosită și trece în starea "idle", caracterizată prin $ALE=0$ și $\overline{RD}=\overline{WR}=1$.

A doua situație este cea specifică tratării solicitărilor de întrerupere pe liniile TRAP, RST 6.5 sau RST 5.5, când microprocesorul generează intern codul instrucțiunii RESTART corespunzătoare. În acest caz nu se mai activează semnalul \overline{INTA} , deci magistrala de date nu este utilizată.

Trebuie remarcat faptul că în ciclurile de tip BI nu se ia în considerare semnalul READY, neputând fi introduse stări de așteptare.

4.2.2. Structura sistemelor cu 8085A realizate cu circuite standard

Pentru realizarea unui sistem cu microprocesor 8085A este necesar să se realizeze demultiplexarea magistralei AD_7+AD_0 , în vederea obținerii octetului inferior al adresei. De asemenea, pentru controlul dispozitivelor de memorie și I/E standard este necesară decodificarea semnalului de comandă IO/\overline{M} , astfel ca împreună cu \overline{RD} și \overline{WR} să se genereze semnalele de comandă \overline{MEMR} , \overline{MEMW} , $\overline{I/OR}$ și $\overline{I/OW}$.

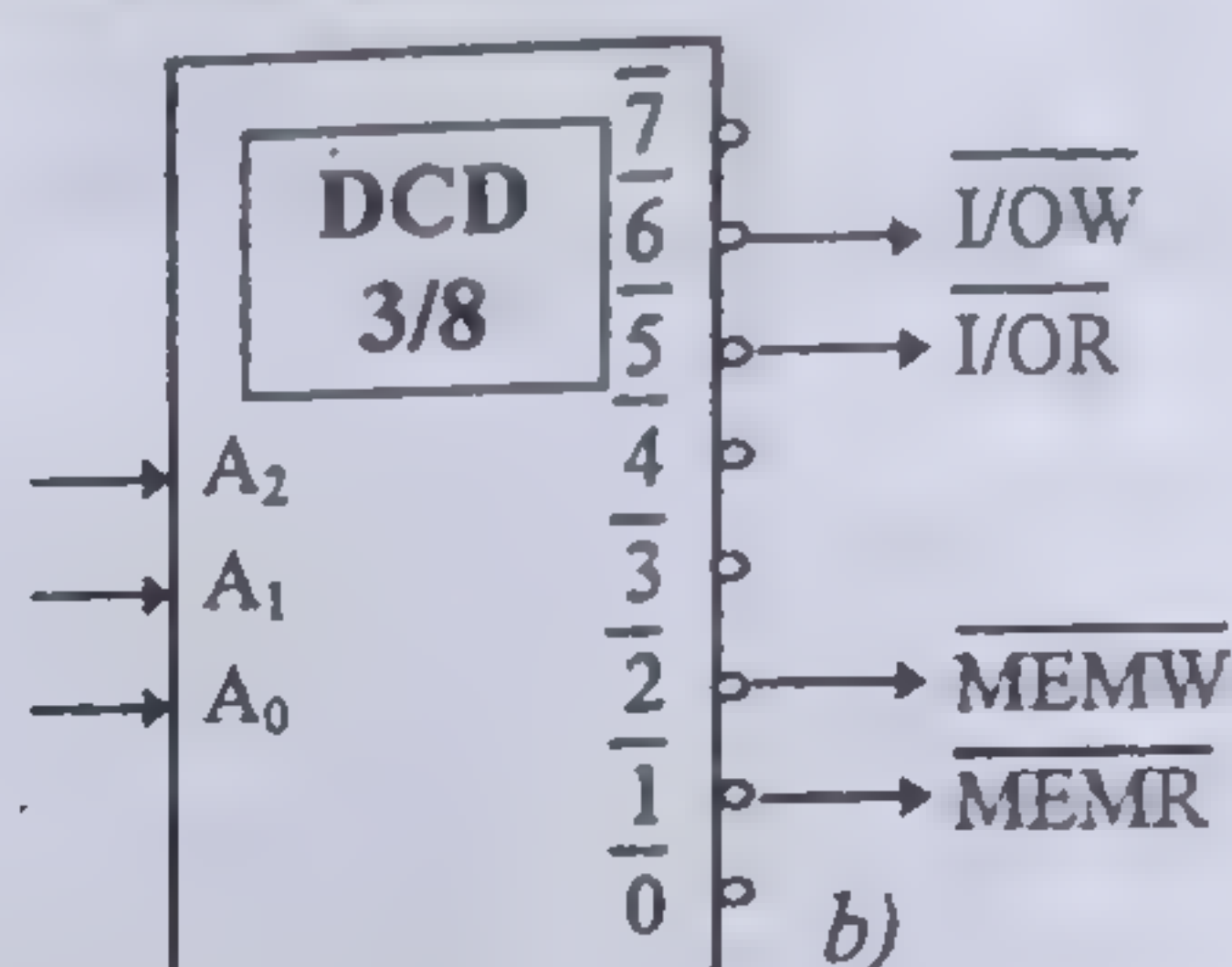
Prima problemă se rezolvă prin folosirea unui circuit 8212, controlat pe intrarea STB de semnalul ALE și care să funcționeze în regimurile DATA IN - DATA LATCH (v.fig.1.25b).

Pentru conservarea facilității TS a liniilor A_7+A_0 este necesar ca 8212 să fie controlat de semnalul HLDA, pe linia $\overline{DS1}$.

Cea de-a doua problemă necesită implementarea tabelului de adevăr din fig.4.14a, ușor de realizat cu ajutorul unui decodificator standard 3/8, așa cum se arată în fig.4.14b.

$\overline{IO/M}$	\overline{RD}	\overline{WR}	Semnale de comandă
0	0	1	\overline{MEMR}
0	1	0	\overline{MEMW}
1	0	1	$\overline{I/OR}$
1	1	0	$\overline{I/OW}$

a)



b)

Fig.4.14. Generarea magistralei de comandă standard

În fig.4.15 se prezintă structura unui sistem cu 8085A care are în componență dispozitive de memorie și circuite I/E standard, de tipul celor utilizate și în sistemele cu 8080A (v.fig.4.9): 8251A (interfață serială), 8253-5 (timer programabil), 8255A-5 (interfață paralelă), 8257-5 (controler DMA), 8259A-5 (controler de întreruperi).

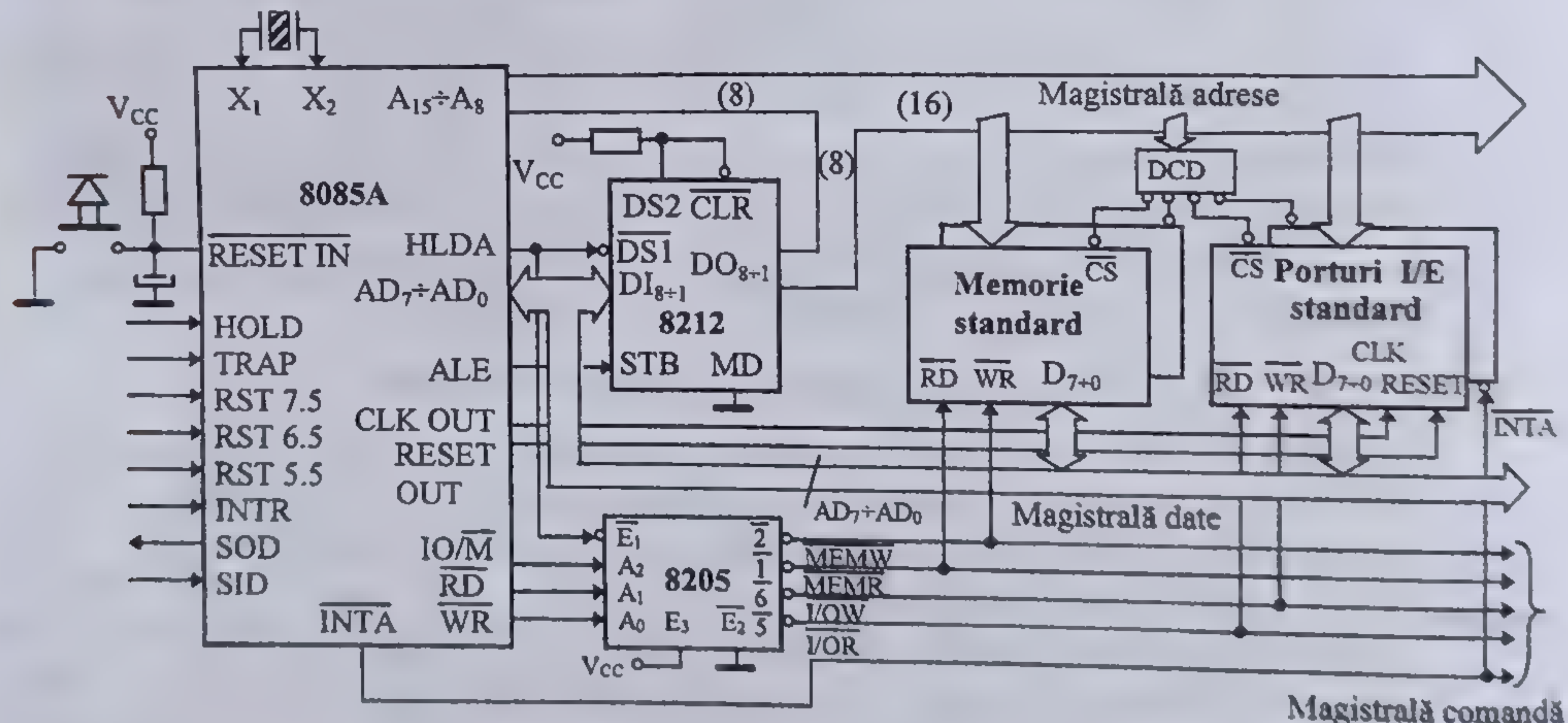
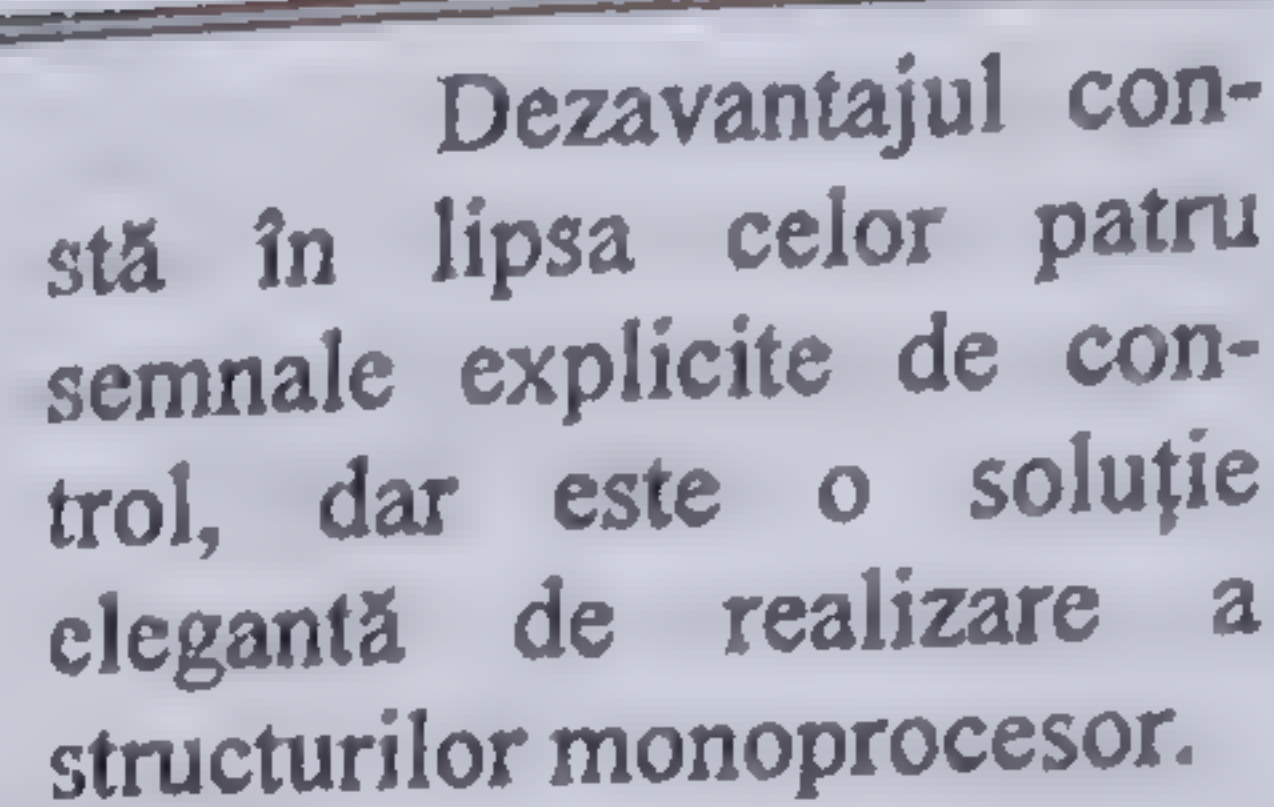


Fig.4.15. Structura unui sistem cu 8085A

Pentru generarea magistralei de comandă s-a folosit un circuit 8205, la care una din intrările de validare (E - Enable) este comandată de semnalul HLDA. În acest mod, în cazul regimului de cedare de magistrale (HOLD = 1), se asigură trecerea în starea de înaltă impedanță a magistralei de comandă.

Pentru controlul dispozitivelor de memorie și a porturilor I/E poate fi utilizată și o a doua soluție, prin folosirea a două decodificatoare distincte pentru cele două tipuri de dispozitive. Decodificatoarele trebuie să aibă intrări de activare (E și \overline{E}), care sunt comandate de semnalul $\overline{IO/M}$ (fig.4.16).



Funcționarea este evidentă: când 8085A dialoghează cu memoria, este activat DCD pentru memorie ($IO / \overline{M} = 0$); când se referă la un port I/E, este activat numai DCD I/E ($IO / \overline{M} = 1$). În ambele cazuri, sensul de transfer pe magistrala de date este stabilit de semnalele \overline{RD} și \overline{WR} .

Fig.4.16. Selecția cu decodificare

Odată cu microprocesorul 8085A, firma Intel a produs încă două circuite: **8155/56** și respectiv **8355/8755**, pentru realizarea de sisteme cu un număr minim de componente, fără a fi necesare dispozitive auxiliare de demultiplexare și decodificare.

Circuitul 8155 este un circuit programabil complex, care conține o memorie RAM static de 256 de octeți, 22 de linii I/E sub forma a trei porturi: A și B de 8 biți, C de 6 biți, precum și un ceas programabil (timer) de 16 biți. În fig.4.17a este prezentată schema bloc simplificată a acestui circuit.

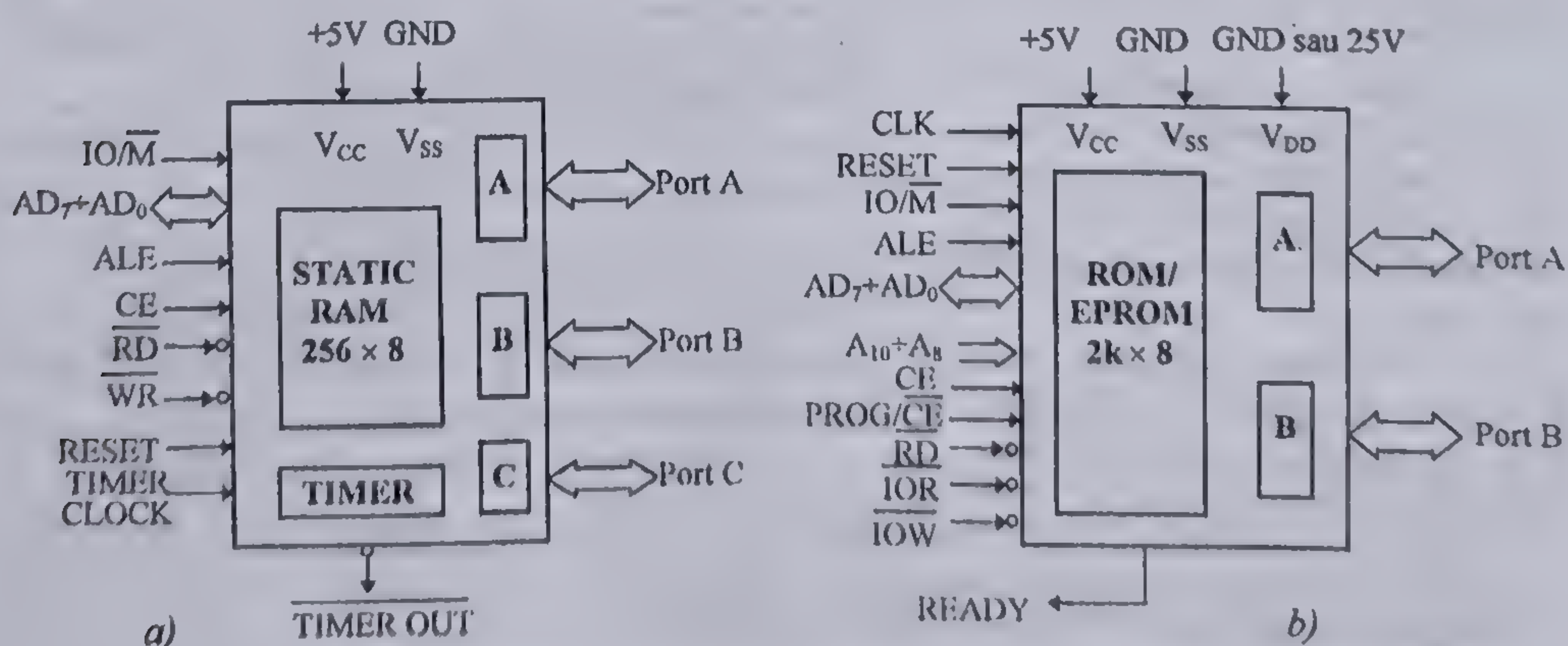
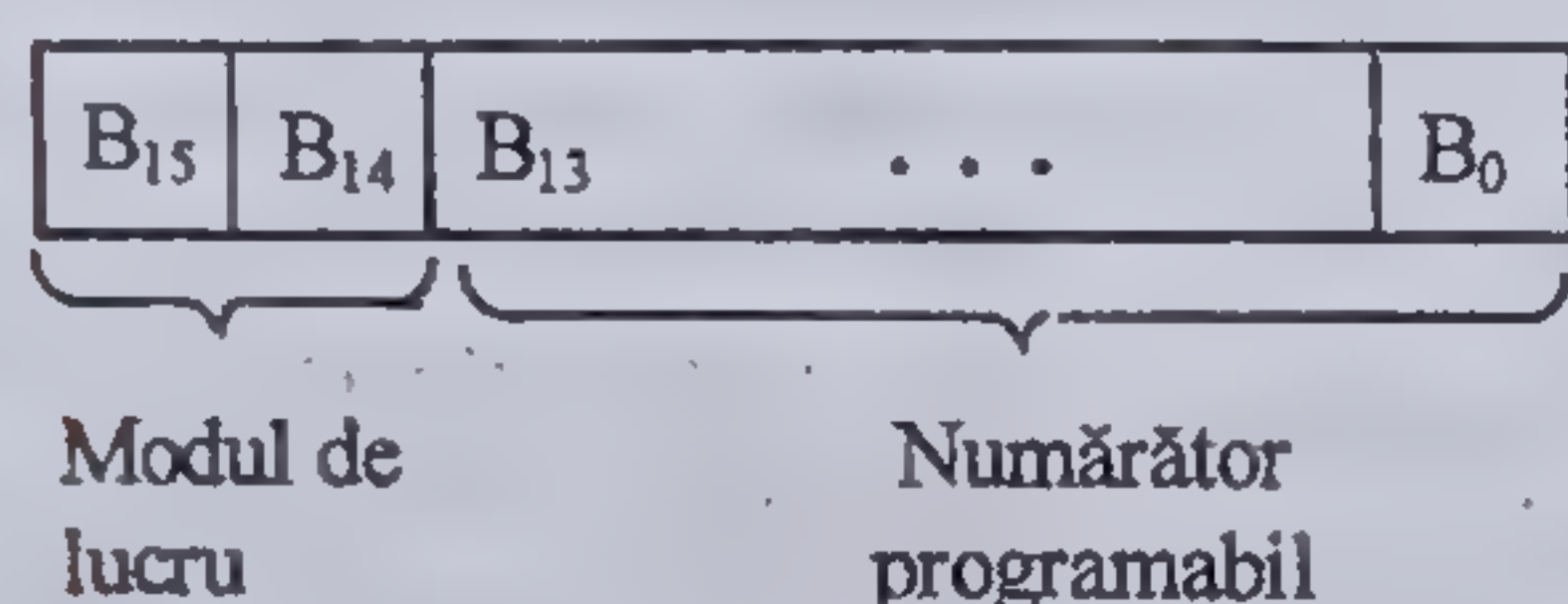


Fig.4.17. Schema bloc a circuitelor dedicate 8155 și 8355/8755

Dispozitivul posedă toate semnalele necesare conectării directe la 8085A, fără nici o structură logică suplimentară. La activarea semnalului RESET, cele trei porturi se configurează

implicit ca porturi de intrare. Ulterior, toate pot fi programate ca porturi de intrare sau de ieșire fără confirmare; porturile A și B pot funcționa și în modul sincronizat, cu sau fără întreruperi, când liniile portului C se utilizează pentru comanda transferului cu confirmare (hardware handshaking). Practic, această structură înglobează un circuit de interfață paralelă de tipul 8255A puțin simplificat, care poate fi utilizat ca interfață paralelă cu perifericele (v. §4.7.1). Prin intermediul semnalului IO/\overline{M} se face distincție între dialogul cu memoria ($IO/\overline{M}=0$) și cel cu porturile I/E ($IO/\overline{M}=1$). Octeții transmiși de microprocesorul 8085A pe magistrala $AD_7 \div AD_0$ pot fi adrese, care sunt reținute în exterior în momentul activării semnalului ALE, sau pot fi date ori comenzi pentru porturi.

Timerul folosește pentru contorizare ultimii 14 biți, primii doi biți mai semnificativi fiind utilizați pentru programarea modului de lucru (fig.4.18).



a)

B ₁₅	B ₁₄	Mod de operare
0	0	Numărător de evenimente
0	1	Generator de undă dreptunghiulară
1	0	Monostabil
1	1	Generator de impulsuri

b)

Fig.4.18. Formatul cuvântului la timer (a) și modurile de lucru (b)

Evenimentele externe sunt convertite în semnale care se aplică pe intrarea TIMER CLOCK și asigură decrementarea valorii programate în numărător. Astfel, este posibilă implementarea prin hardware a ceasurilor și a numărătoarelor de evenimente. În cazul sistemelor mai complexe, se folosesc dispozitive specializate cu mai multe canale (v. §4.5).

Pentru programarea dispozitivelor I/E integrate pe cip, 8155 este prevăzut cu un registru de comandă/stare, pentru care semnificația biților este prezentată în fig.4.19a.

Se observă că porturile A și B pot fi programate independent, ca porturi de intrare sau de ieșire. În schimb, portul C este considerat ca fiind format din două semiporturi de 3 linii (PC_0+PC_2 , respectiv PC_3+PC_5), care pot fi controlate în 4 alternative distincte: ALT1 - IN; ALT2 - OUT; ALT3 - PC_0+PC_2 - linii de confirmare pentru portul A, PC_3+PC_5 - OUT; ALT4 - linii de confirmare.

În modul cu confirmare, porturile A și B pot cere întreruperi prin PC_0 (port A) și PC_3 (port B), dacă IEA și respectiv IEB sunt în "1" logic. Liniile PC_1 și PC_4 sunt ieșiri folosite

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
TM2	TM1	IEB	IEA	PC2	PC1	PB	PA
Comenzi timer: 00 - NOP 01 - STOP 10 - STOP after TC 11 - START		Port B, A - activare întreruperi		Port C 00 - ALT1 01 - ALT2 10 - ALT3 11 - ALT4		Port B, A 0 - Intrare 1 - Ieșire	

a)

b)

Adresa portului I/E			Semnificația portului I/E
A ₇ ÷ A ₃	A ₂	A ₁ A ₀	
x	0	0 0	Registru de comandă/stare
x	0	0 1	Port A
x	0	1 0	Port B
x	0	1 1	Port C
x	1	0 0	Registru timer (LSB)
x	1	0 1	Registru timer (MSB)

Fig.4.19. Formatul cuvântului de comandă (a) și adresele registrelor interne ale circuitului 8155 (b)

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

pentru confirmarea stării tampoanelor porturilor A și respectiv B (BF - Buffer Full), iar PC_2 și PC_5 sunt intrări de strobare, active pe nivel coborât (\overline{STB}).

Biții D_6 și D_7 ai cuvântului de comandă permit transmiterea de comenzi timerului:

- NOP - nici un efect;
- STOP - oprirea numărării;
- STOP AFTER TC - stoparea funcționării după terminarea contorizării;
- START - încărcare contor și declanșare funcționare.

Dacă se programează un nou mod și o nouă constantă de timp în timpul funcționării, noua valoare va fi luată în considerare abia după terminarea contorizării (after TC).

Adresele pentru registrul de comandă/stare, cele trei porturi și pentru timer sunt date în fig.4.19b. Ele sunt reținute de 8155 într-un mod similar cu adresele pentru RAM, cu $ALE = 1$, dar cu $IO/\overline{M} = 1$. Apoi, microprocesorul poate transmite ($\overline{WR} = 0$) sau poate citi ($\overline{RD} = 0$) date din porturi și timer. Deși conținutul timerului poate fi citit direct, o informație corectă se poate obține numai cu o comandă STOP, urmată de citirea conținutului, de reîncărcare și de continuarea numărării.

Registrul de comandă poate fi citit printr-o procedură normală ($\overline{RD} = 0$), informația obținută referindu-se atât la starea transferului prin porturile A și B, cât și la cea a timerului. Din acest motiv, registrul este denumit de comandă/stare, iar formatul cuvântului depus pe liniile $AD_7 \div AD_0$ este cel din fig.4.20.

Bitul D_6 se setează când numărarea s-a încheiat (Terminal Count) și este resetat în momentul citirii stării de către microprocesor sau dacă se începe o nouă contorizare.

Firma Intel produce și circuitul 8156, având facilități identice cu 8155, dar selectabil pe nivel coborât (\overline{CE}).

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
-	TIMER	INTE _B	BF _B	INTR _B	INTE _A	BF _A	INTR _A
Cerere întrerupere de la timer							Cerere de întrerupere Port A
							Tampon Port A plin/gol (intrare/ieșire)
Întreruperi de la portul B validate					Întreruperi de la portul A validate		
					Cerere de întrerupere Port B		
		Tampon Port B plin/gol (intrare/ieșire)					

Fig.4.20. Formatul cuvântului de stare la circuitul 8155

Circuitele 8355/8755 conțin o memorie ROM/EPROM de 2048 octeți și două porturi I/E de 8 biți (fig.4.17b). Adresarea memoriei se face prin liniile $AD_7 \div AD_0$ și $A_{10} \div A_8$. Când 8085A activează linia ALE, starea magistralei $AD_7 \div AD_0$ și a liniilor IO/\overline{M} , $A_{10} \div A_8$, \overline{CE} și \overline{CE} ($PROG/\overline{CE}$ la 8755) este memorată în circuitele 8355/8755. Data din memorie este depusă pe magistrala $AD_7 \div AD_0$ când $IO/\overline{M} = 0$ și $\overline{RD} = 0$. La dispozitivul 8755, programarea memoriei EPROM interne se face pentru fiecare octet, prin depunerea acestuia pe liniile $AD_7 \div AD_0$, simultan cu aplicarea pe linia $PROG/\overline{CE}$ a unui impuls logic "1" (TTL), cu durata de 45ms. Pe durata programării, V_{DD} se conectează la +25V. În timpul funcționării normale, V_{DD} se conectează la masă (GND).

Tab.4.5.

AD_1	AD_0	Port I/E selectat
0	0	Port A
0	1	Port B
1	0	Registrul de direcție al portului A (DDR _A)
1	1	Registrul de direcție al portului B (DDR _B)

Porturile A și B sunt selectate prin intermediul liniilor AD_1 și AD_0 , în conformitate cu tab.4.5. Se observă că fiecare port are un *registru de direcție* (Data Direction Register), prin intermediul căruia se precizează tipul fiecărui pin: valoarea "0" specifică un pin de intrare, iar valoarea "1" un pin de ieșire. Aceste registre nu pot fi citite de microprocesor, dar sunt încărcate cu 00h la resetare.

Înscrierea într-un port se realizează prin activarea de către microprocesor a liniei \overline{IOW} , ceea ce determină memorarea datei prezente pe magistrala $AD_7 \div AD_0$ în portul de ieșire selectat. Starea liniei IO/\overline{M} este ignorată în acest caz. Citirea unui port se poate face fie activând linia \overline{IOR} , fie prin $IO/\overline{M} = 1$ și $\overline{RD} = 0$, caz în care linia \overline{IOR} se conectează la V_{CC} .

Pentru sincronizarea funcționării circuitului cu microprocesorul 8085A se folosesc intrarea CLK și ieșirea READY, care informează UCP despre situațiile în care memoria dispozitivelor 8355/8755 nu are încă datele valide (depuse pe magistrala $AD_7 \div AD_0$). Acest lucru nu afectează reținerea adresei: linia ALE rămâne pe nivel logic "1" cât timp semnalul READY este forțat în "0" și revine pe nivel logic "0" după ce $READY=1$, pe următorul front ascendent al semnalului de tact (pe linia CLK).

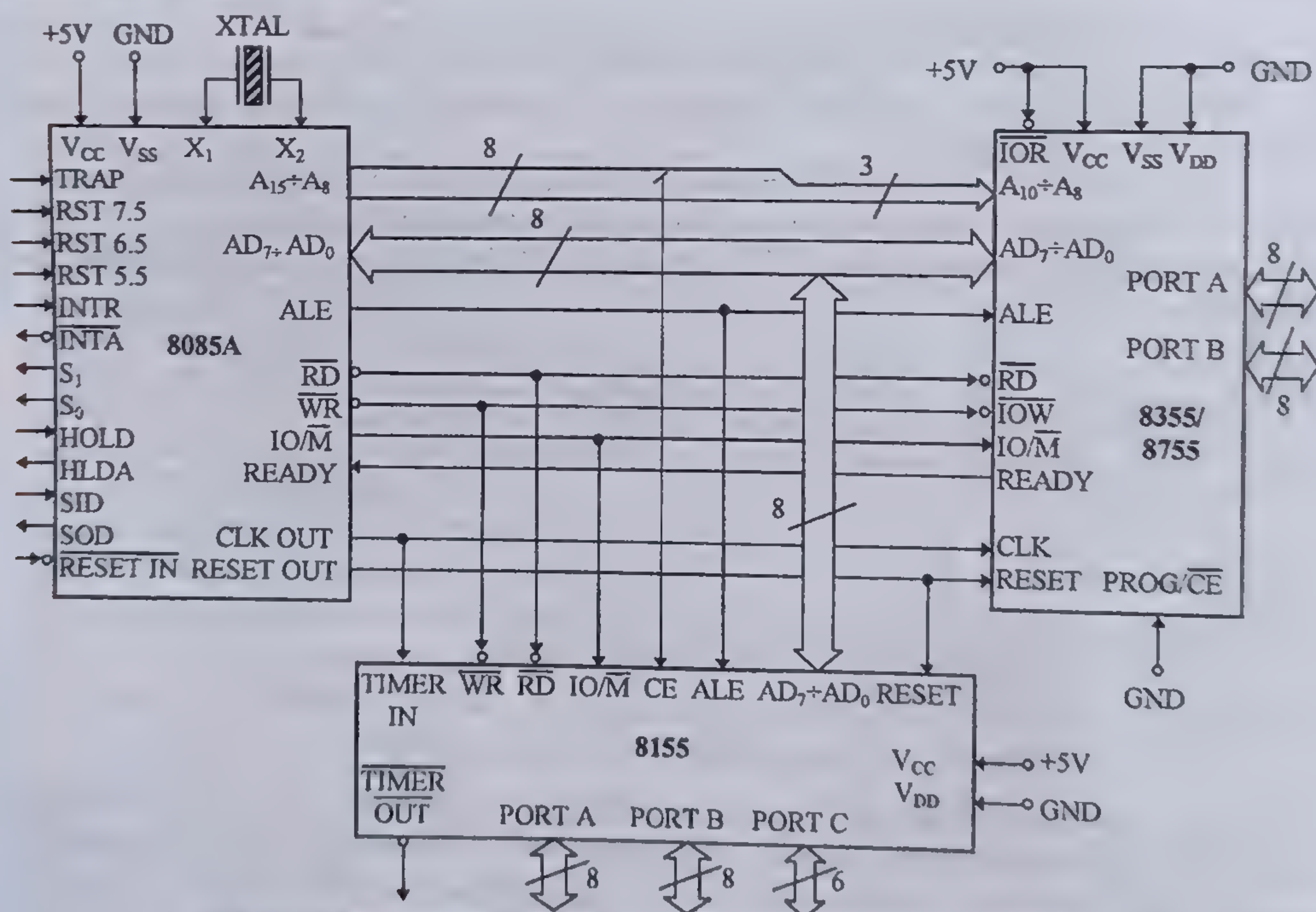


Fig.4.21. Structura minimală a unui sistem cu microprocesor 8085A

Cu ajutorul celor două tipuri de dispozitive poate fi implementată o structură de sistem minimală și performantă, formată numai din trei circuite LSI: 8085A, 8155/8156 și 8355/8755. În fig.4.21 se prezintă un astfel de microsistem, atractiv atât pentru aplicații de mică complexitate cât și pentru sisteme distribuite de conducere automată.

Atât secțiunea de memorie, cât și cea de I/E se pot extinde ușor prin conectarea de dispozitive 8155 sau 8755 direct pe magistralele microprocesorului. Această soluție - de integrare pe un singur cip a cât mai multor facilități - prefigurează evoluția ulterioară către microcontrolere pe 8 biți, tendință care se înregistrează și în prezent, la toți marii producători de microprocesoare.

4.3. Programarea microprocesoarelor 8080A și 8085A

Realizarea aplicațiilor cu aceste două tipuri de microprocesoare impune cunoașterea resurselor software: formatul instrucțiunilor, modurile de adresare și setul de instrucțiuni. În această privință, așa cum s-a menționat anterior, microprocesorul 8085A este complet compatibil în jos cu 8080A, având în plus doar două instrucțiuni, pentru lucrul cu liniile suplimentare de întrerupere și de transfer serial (SID/SOD). De aceea, cu excepția cazurilor în care se specifică altfel, informațiile prezentate în acest paragraf sunt valabile pentru ambele microprocesoare.

4.3.1. Formatul instrucțiunilor

În corespondență cu dimensiunea cuvântului pe care-l procesează, instrucțiunile sunt organizate în cuvinte de 8 biți și pot avea, în funcție de complexitate, între 1 și 3 octeți. Indiferent de lungimea instrucțiunii, primul octet conține întotdeauna codul generalizat (OPCODE), iar următorii sunt operanzi sau adrese de operanzi, de 8 sau 16 biți (fig.4.22).

Bitul din extrema dreaptă al unui octet este considerat a fi cel mai puțin semnificativ (lsb - least significant bit), iar cel din extrema stângă - cel mai semnificativ (msb - most significant bit). Octeții unei instrucțiuni se amplasează în memorie în ordinea specificată în fig.4.22, în sensul crescător al adreselor. Operanzii de 8 biți pot fi constante sau adrese de port I/E, iar operanzii de 16 biți - constante sau adrese de memorie.

4.3.2. Moduri de adresare

După cum s-a văzut în capitolul 1, modurile de adresare caracterizează eficiența unui microprocesor în ceea ce privește specificarea adresei unui operand. Microprocesoarele 8080/8085 folosesc 6 moduri de adresare. În acest paragraf, referirile la magistrala de adrese ca $A_{15}+A_8$ și A_7+A_0 se fac pentru microprocesorul 8080; ele rămân valabile și pentru 8085, dacă prin A_7+A_0 se înțelege octetul inferior al adresei, depus pe liniile AD_7+AD_0 și memorat într-un latch extern, la activarea semnalului ALE.

Adresarea implicită (cu registru specific). În cazul în care operandul se află într-unul din registrele interne speciale ale microprocesorului (A, PC, SP) sau în fanionul CY, opcodul instrucțiunii indică implicit localizarea operandului.

Exemple:

ADD B - (A) ← (A) + (B) - conținutul registrului B se adună la cel al acumulatorului; acumulatorul este adresat implicit;

STC - setarea fanionului CY, adresat implicit.

Avantajele acestui mod de adresare sunt reducerea dimensiunii codului și a duratei de execuție, întrucât microprocesorul localizează operandul încă din faza de decodificare a instrucțiunii.

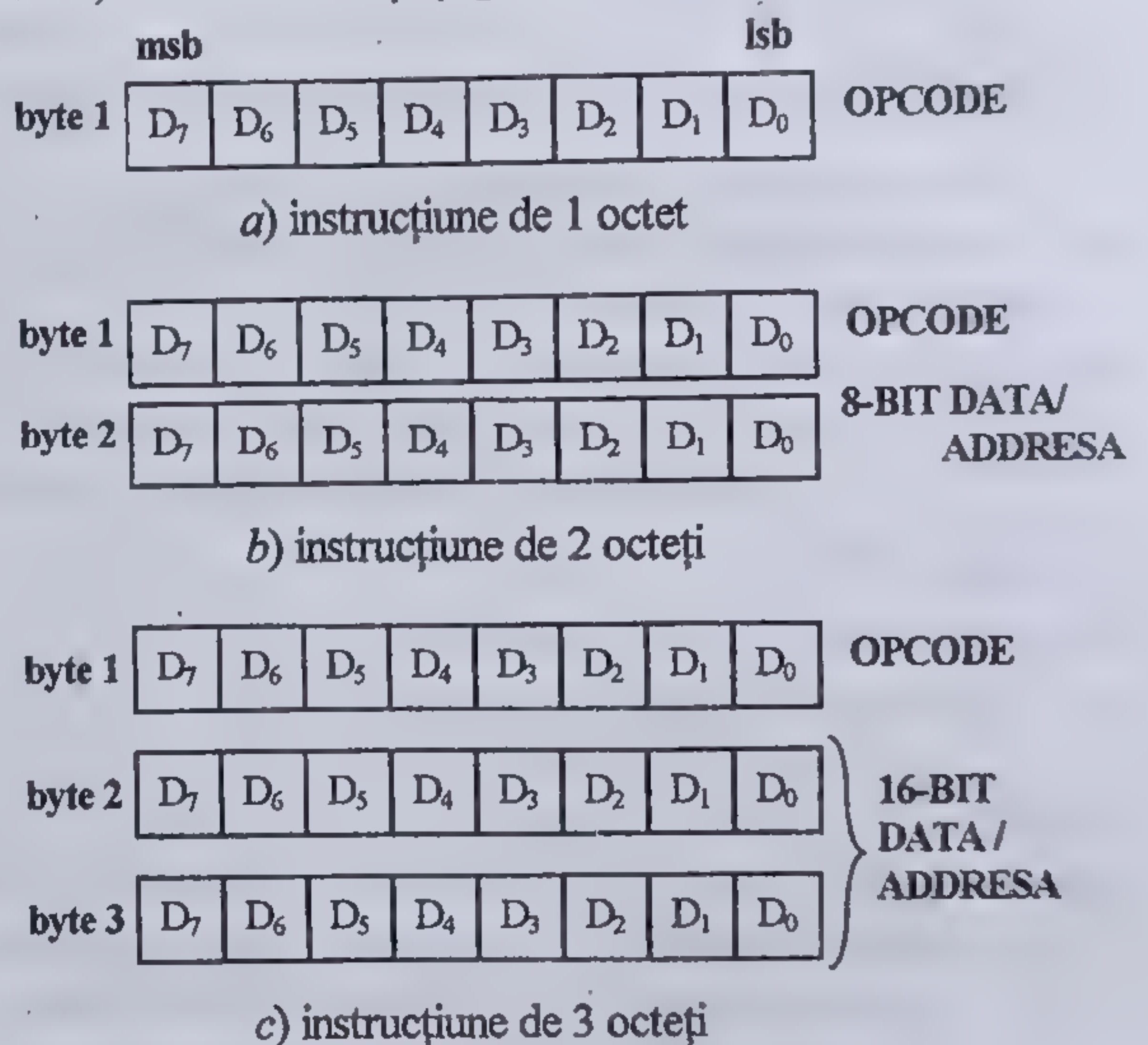
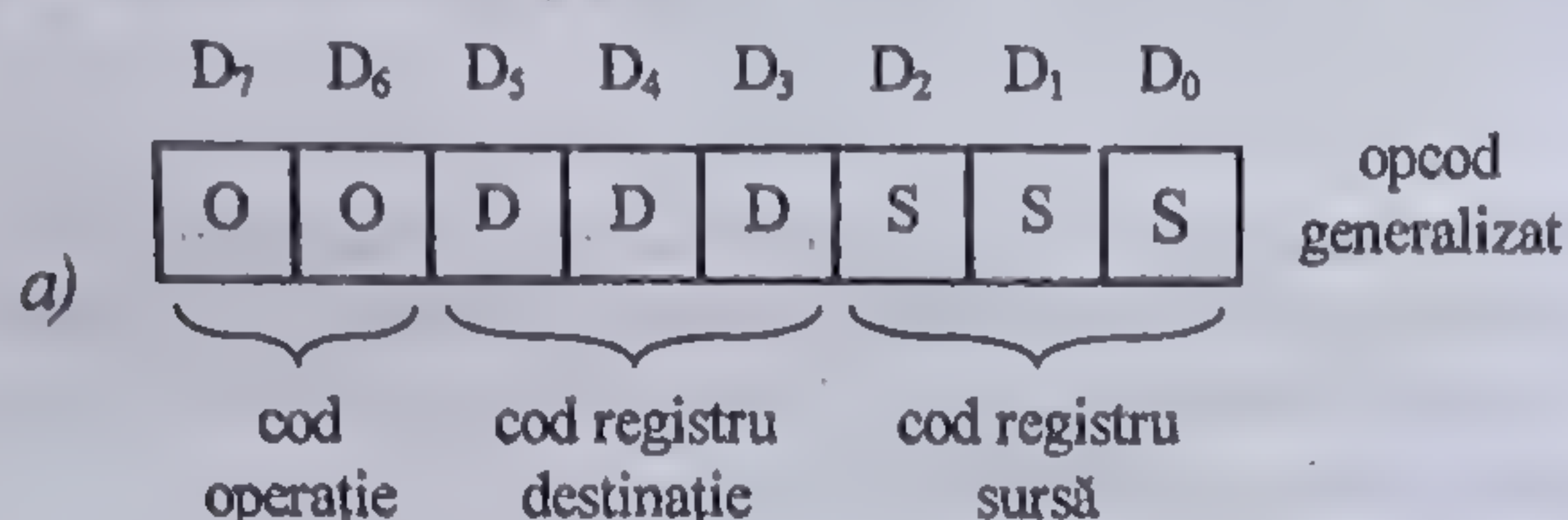


Fig.4.22. Formatul instrucțiunilor la μP 8080/8085

Adresarea explicită. Teoretic, adresarea implicită s-ar putea referi la oricare din registrele de lucru ale microprocesorului, dar acest fapt ar fi condus la creșterea complexității blocului de decodificare a instrucțiunilor. Din acest motiv, pentru registrele interne cu destinație generală s-a apelat la *adresarea explicită a registrelor*, denumită și *adresarea registrelor* sau *adresarea la registru*.

Acest mod de adresare se folosește atunci când operandul instrucțiunii se află în unul din registrele A, B, C, D, E, H sau L. Instrucțiunile care folosesc adresarea la registru sunt de un octet și au structura din fig.4.23a.



b)

DDD sau SSS	Registru
0 0 0	B
0 0 1	C
0 1 0	D
0 1 1	E
1 0 0	H
1 0 1	L
1 1 0	M (*)
1 1 1	A

Fig.4.23. Formatul instrucțiunilor cu adresare explicită

În afară de codul operației (primii doi biți mai semnificativi - OO), opcodul generalizat mai conține două grupuri de 3 biți, care codifică registrele destinație (DDD) și sursă (SSS) utilizate de instrucțiune. În tabelul din fig.4.23b se dau codurile aferente registrelor interne ale microprocesorului. După cum se observă, combinația "1 1 0" nu corespunde nici unui registru, aceasta fiind rezervată de producător pentru referirea la conținutul unei locații de memorie (*). Modul de adresare la registru este utilizat, de exemplu, de grupul instrucțiunilor de transfer între registre:

Exemplu:

MOV $r_1, r_2 - (r_1) \leftarrow (r_2)$ - conținutul registrului r_2 se transferă (prin copiere) în registrul r_1 , unde r_1 și r_2 pot fi oricare din registrele A, B, C, D, E, H și L; conținutul lui r_2 nu se modifică.

Ca și adresarea implicită, nici cea explicită nu reclamă cicluri mașină suplimentare, în afară de ciclul FETCH, întrucât operandii se află deja în interiorul microprocesorului, în registrele interne. Se păstrează deci toate avantajele prezentate la modul de adresare implicită, la care se mai adaugă o flexibilitate sporită, întrucât nu mai există limitarea la o singură localizare a operandului. Astfel, o aceeași operație se poate face cu regiștri diferiți.

Adresarea imediată. În acest caz, instrucțiunea conține operandul (unul sau doi octeți), dispus în memorie *imediat* după codul operației (v.fig.4.22b, c). Adresa operandului se formează prin incrementarea numărătorului de program (PC) și se depune pe magistrala de adrese. Instrucțiunile care utilizează acest mod de adresare au 2 sau 3 octeți și se folosesc la inițializarea registrelor simple sau pereche, cu valori constante de 8, respectiv 16 biți.

Exemple:

MVI A,data8 - $(A) \leftarrow (\text{byte } 2)$ - valoarea celui de al 2-lea octet al instrucțiunii (byte 2 = data8) se încarcă în registrul A.

LXI H,data16 - $(L) \leftarrow (\text{byte } 2), (H) \leftarrow (\text{byte } 3)$, unde data16 = (byte 3)(byte 2) - valoarea celui de al 2-lea octet al instrucțiunii se încarcă în registrul L, iar a celui de-al 3-lea octet, în registrul H; cu alte cuvinte, în registrul pereche H (aici H desemnează perechea de registre H-L) se încarcă constanta de 16 biți data16.

Spre deosebire de modurile de adresare anterioare, crește dimensiunea codului cu 1, respectiv 2 octeți și, în mod corespunzător, crește și durata de execuție a instrucțiunilor. Acestea

folosesc, pe lângă ciclul FETCH - obligatoriu, încă 1 sau 2 cicluri de citire memorie, necesare pentru aducerea operandului în registrele interne ale microprocesorului.

Adresarea directă. Atunci când se referă la adresarea operanzilor din memorie, octeții 2 și 3 ai instrucțiunii conțin adresa locației de memorie în care este amplasat operandul. Este cel mai simplu mod de adresare a variabilelor din memorie de 1 sau 2 octeți, dar consumă un timp mai lung, chiar și decât adresarea imediată. Aceasta deoarece instrucțiunile care utilizează adresarea directă nu conțin operandul, ci doar adresa acestuia. Astfel, după ciclul FETCH și cei doi cicluri de citire memorie necesare pentru a aduce adresa în registrele W-Z, urmează un nou ciclu de acces la memorie, în care are loc accesul la operand. În acest ciclu, conținutul registrelor W-Z este depus pe magistrala de adrese și operandul este adus din memorie într-unul din registrele interne ale microprocesorului, sau este depus de către acesta în memorie, la adresa respectivă.

Exemple:

LDA *addr* - $(A) \leftarrow ((\text{byte } 3)(\text{byte } 2))$ - conținutul locației de memorie a cărei adresă este specificată în octeții 2 și 3 ai instrucțiunii prin *addr* = (byte 3)(byte 2), se încarcă în acumulator. Execuția acestei instrucțiuni durează 13 stări, organizate în 4 cicluri mașină.

SHLD *addr* - $((\text{byte } 3)(\text{byte } 2)) \leftarrow (L)$, $((\text{byte } 3)(\text{byte } 2) + 1) \leftarrow (H)$ - conținutul registrului pereche H se depune în memorie, la adresa specificată prin octeții 2 și 3 ai instrucțiunii. Memorarea se face în locații succesive, în sensul crescător al adreselor, fiind înscris mai întâi (L) și apoi (H). Instrucțiunea se execută în 16 stări, grupate în 5 cicluri mașină.

Se observă că adresele locațiilor de memorie sunt constante. O astfel de adresare este posibilă pentru referirea la variabile de memorie alocate static, la adrese cunoscute la momentul scrierii programului și care nu se modifică pe parcursul execuției acestuia.

Adresarea directă se utilizează și în cazul dialogului cu dispozitivele I/E. Dar, întrucât adresa acestora are lungimea de numai 1 octet, atât dimensiunea cât și durata execuției instrucțiunilor respective se reduc în mod corespunzător.

Exemple:

IN *port* - $(A) \leftarrow (\text{byte } 2)$, unde *port* = byte 2 - se încarcă în A octetul depus pe magistrala de date de către portul a cărui adresă este dată de cel de-al doilea octet al instrucțiunii.

OUT *port* - $(\text{byte } 2) \leftarrow (A)$, cu *port* = byte 2 - se încarcă conținutul acumulatorului în portul cu adresa specificată de cel de-al doilea octet al instrucțiunii.

Aceste două instrucțiuni sunt de fapt singurele pe care le posedă 8080/8085 și servesc pentru lucrul cu maximum 256 porturi de intrare și 256 de ieșire.

Adresa portului este de 8 biți și este încărcată, în cel de-al doilea ciclu al instrucțiunii (după FETCH), în ambele registre, W și Z. Conținutul acestor două registre se depune, ca și în cazul adresării memoriei, pe liniile $A_{15} \div A_0$. De aici rezultă că porturile I/E pot fi adresate atât pe liniile inferioare, $A_7 \div A_0$, cât și pe liniile superioare ale magistralei de adresă, $A_{15} \div A_8$. Întrucât liniile inferioare sunt folosite și pentru adresarea memoriei, pentru adresarea porturilor se preferă uneori partea superioară a magistralei de adrese, din considerente de încărcare a acesteia (fan-out).

Adresarea indirectă sau cu registru pereche - se caracterizează prin faptul că adresa operandului din memorie se află deja în interiorul microprocesorului, într-un registru pereche. Va rezulta deci o reducere substanțială atât a dimensiunii, cât și a duratei de execuție a instrucțiunilor cu adresare indirectă. Astfel, după ciclul FETCH se poate trece imediat la ciclul de acces la operandul din memorie. Conținutul registrului pereche de adresare indirectă este depus pe magistrala de adrese $A_{15} \div A_0$ și operandul este citit sau înscris, în funcție de tipul instrucțiunii.

Un alt avantaj al acestui mod de adresare este acela că permite referirea la operanzi din memorie a căror adresă nu este cunoscută la momentul scrierii programului, ci se determină la momentul execuției acestuia, permițând lucrul cu variabile alocate dinamic în spațiul de memorie.

Cele mai multe instrucțiuni cu adresare indirectă folosesc registrul pereche H (format din registrele simple H și L). Referirea indirectă la un octet din memorie cu registrul pereche H este desemnată în instrucțiuni prin notația "M". Microprocesorul află despre localizarea indirectă a operandului imediat după decodificarea opcodului instrucțiunii, care conține în câmpul SSS sau DDD combinația "1 1 0", semnalată la adresarea registrelor, în tab.4.23b.

Exemple:

MOV A,M - $(A) \leftarrow ((H)(L))$ - se transferă în registrul A conținutul locației de memorie a cărei adresă se află deja încărcată în registrul pereche H (de exemplu, printr-o instrucțiune cu adresare imediată - LXI H,addr).

Nu numai registrul pereche H poate fi utilizat la adresarea indirectă, ci și registrele pereche B și respectiv D, desemnate prin *rp* în exemplele următoare:

LDAX *rp* - $(A) \leftarrow ((rp))$ - încarcă în acumulator octetul de la adresa specificată indirect, de conținutul din acel moment al registrului pereche *rp*.

STAX *rp* - $((rp)) \leftarrow (A)$ - înscrie conținutul acumulatorului în locația de memorie indicată, în mod indirect, de conținutul registrului pereche *rp*.

O altă posibilitate de adresare indirectă a unei locații de memorie o constituie folosirea registrului indicator al vârfului stivei, SP. În acest caz, 8080 are două tipuri de instrucțiuni, cu mnemonicele PUSH și POP, care asigură salvarea și respectiv refacerea registrelor pereche PSW (acumulatorul și indicatorii de condiții), B, D, și H în/din stivă. Adresa vârfului stivei, organizată întotdeauna în memoria RAM, se află în SP; acesta se decrementează/incrementează automat la fiecare astfel de transfer.

Adresarea combinată se referă la utilizarea mai multor moduri de adresare, dintre cele prezentate mai sus, în cadrul aceleiași instrucțiuni. Există astfel combinații de adresare implicită cu adresare directă (LDA addr, exemplificată deja la adresarea directă), adresare explicită cu adresare indirectă (MOV A,M - prezentată deja la adresarea indirectă), sau adresare directă cu adresare indirectă și implicită, așa cum se arată în exemplele de mai jos.

Exemple:

CALL addr - adresa specificată în instrucțiune pentru a indica punctul de intrare într-o subrutină este precedată de salvarea în stivă a conținutului contorului de program (PC), realizată printr-o adresare indirectă, implicit cu registrul SP.

RST *n* - apelul subrutinei de la adresa egală cu $8 \times n$, în care $n = 0 \div 7$ - instrucțiunea este similară cu cea precedentă, cu deosebirea că are un singur octet, iar numărul de adrese de apel este limitat la 8 (0000h, 0008h, 0010h, ..., 0038h).

4.3.3. Setul de instrucțiuni

După cum s-a văzut în §4.3.1, toate instrucțiunile microprocesoarelor 8080/8085 au codul operației pe un singur octet. Din totalul celor 2^8 combinații posibile se folosesc numai 244/246 pentru instrucțiunile executabile.

Instrucțiunile unui microprocesor se pot clasifica după mai multe criterii: mnemonica operației, mnemonica instrucțiunii (care include și operanzii), funcționalitate etc. Dacă setul de instrucțiuni se clasifică după mnemonicele operațiilor, rezultă 78/80 de tipuri de instrucțiuni. În cazul în care se ține cont de mnemonicele instrucțiunilor, rezultă 72/74 de tipuri de instrucțiuni.

Cea mai importantă pentru utilizator este clasificarea setului de instrucțiuni după criteriul funcționalității. În cazul microprocesoarelor 8080/8085 se pot distinge 5 categorii de instrucțiuni, și anume:

- instrucțiuni de transfer, care realizează copierea datelor între registre sau între registre și memorie; aceste instrucțiuni nu afectează fanioanele de condiție.

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

- b) instrucțiuni aritmetice, care realizează operații aritmetice de adunare și scădere cu cuvinte din registre sau din memorie. Instrucțiunile din această categorie afectează toate fanioanele.
- c) instrucțiuni logice, care realizează operațiile logice de bază asupra cuvintelor binare. Toți indicatorii de condiție sunt afectați (CY și AC sunt resetați automat după o astfel de instrucțiune).
- d) instrucțiuni de ramificare, care permit schimbarea succesiunii uniforme de execuție a instrucțiunilor și lucrul cu subrutine. Ramificările pot fi *necondiționate* sau *condiționate* de valorile fanioanelor. În acest scop există 8 condiții distincte referitoare la valorile fanioanelor Z, CY, P și S, specificate în instrucțiuni prin CC:

NZ (Not Zero) - rezultatul ultimei operații aritmetice sau logice este diferit de 0 ($Z=0$);

Z (Zero) - rezultatul este 0 ($Z=1$);

NC (Not Carry) - nu a existat transport ($CY=0$);

C (Carry) - a existat transport ($CY=1$);

PO (Parity Odd) - rezultatul are un număr impar de biți "1" ($P=0$);

PE (Parity Even) - numărul de biți "1" al rezultatului este par ($P=1$);

P (Plus) - ($S=A_7=0$);

M (Minus) - în acumulator se află un număr negativ ($S=A_7=1$).

- e) instrucțiuni de lucru cu stiva, de I/E și de comandă a microprocesorului.

În tab.4.6 este prezentat setul de instrucțiuni al microprocesoarelor 8080/8085. Se precizează mnemonica instrucțiunii, se face o scurtă descriere, se indică modul (modurile) de adresare folosit(e) și cum sunt afectați indicatorii de condiție. În ultima coloană sunt specificate numărul de cicluri mașină și numărul de stări necesare execuției; în paranteză este dat numărul de stări de la 8085A.

Tab.4.6.

Nr. crt.	Mnemonica instrucțiunii	Descrierea instrucțiunii		Mod(uri) de adresare	Flaguri afectate	Nr. CM/ Nr. stări
Instrucțiuni de transfer						
1	MOV r_1, r_2	MOVE register to register	$(r_1) \leftarrow (r_2)$	la registru	-	1/5 (4)
2	MOV r, M	MOVE Memory to register	$(r) \leftarrow ((H)(L))$	indirectă + la registru	-	2/7 (7)
3	MOV M, r	MOVE register to Memory	$((H)(L)) \leftarrow (r)$	la registru + indirectă	-	2/7 (7)
4	MVI $r, data8$	MoVe to register Immediate	$(r) \leftarrow data8$	imediată + la registru	-	2/7 (7)
5	MVI $M, data8$	MoVe to Memory Immediate	$((H)(L)) \leftarrow data8$	imediată + indirectă	-	3/10(10)
6	LXI $rp, data16$	Load register pair Immediate	$(r_L) \leftarrow low(data16)$ $(r_H) \leftarrow high(data16)$	imediată + la registru	-	3/10(10)
7	LDA $addr$	LoaD Accumulator direct	$(A) \leftarrow (addr)$	directă + implicită	-	4/13(13)
8	STA $addr$	Store Accumulator direct	$(addr) \leftarrow (A)$	implicită + directă	-	4/13(13)
9	LHLD $addr$	Load H and L Direct	$(L) \leftarrow (addr)$ $(H) \leftarrow (addr+1)$	directă + implicită	-	5/16(16)
10	SHLD $addr$	Store H and L Direct	$(addr) \leftarrow (L)$ $(addr+1) \leftarrow (H)$	implicită + directă	-	5/16(16)

Tab.4.6 (cont.)

11	LDAX <i>rp</i>	LoaD Accumulator indirect	$(A) \leftarrow ((rp)),$ unde $rp = B$ sau D	indirectă + implicită	-	2/7 (7)
12	STAX <i>rp</i>	Store Accumulator indirect	$((rp)) \leftarrow (A),$ unde $rp = B$ sau D	implicită + indirectă	-	2/7 (7)
13	XCHG	eXCHanGe H and L with D and E	$(H) \leftrightarrow (D)$ $(L) \leftrightarrow (E)$	implicită	-	1/4 (4)
Instrucțiuni aritmetice						
14	ADD <i>r</i>	ADD register to accumulator	$(A) \leftarrow (A) + (r)$	implicită + la registru	toate	1/4 (4)
15	ADD M	ADD Memory to accumulator	$(A) \leftarrow (A) + ((H)(L))$	implicită + indirectă	toate	2/7 (7)
16	ADI <i>data8</i>	ADd Immediate to accumulator	$(A) \leftarrow (A) + data8$	implicită + imediată	toate	2/7 (7)
17	ADC <i>r</i>	ADd register to A with Carry	$(A) \leftarrow (A) + (r) + (CY)$	implicită + la registru	toate	1/4 (4)
18	ADC M	ADd memory to A with Carry	$(A) \leftarrow (A) + ((H)(L)) + (CY)$	implicită + indirectă	toate	1/4 (4)
19	ACI <i>data8</i>	Add Immediate to A with Carry	$(A) \leftarrow (A) + data8 + (CY)$	implicită + imediată	toate	2/7 (7)
20	SUB <i>r</i>	SUBtract register from A	$(A) \leftarrow (A) - (r)$	implicită + la registru	toate	1/4 (4)
21	SUB M	SUBtract Memory from A	$(A) \leftarrow (A) - ((H)(L))$	implicită + indirectă	toate	2/7 (7)
22	SUI <i>data8</i>	SUBtract Immediate from A	$(A) \leftarrow (A) - data8$	implicită + imediată	toate	2/7 (7)
23	SBB <i>r</i>	SuBtract register from A with Borrow	$(A) \leftarrow (A) - (r) - (CY)$	implicită + la registru	toate	1/4 (4)
24	SBB M	SuBtract Memory from A with Borrow	$(A) \leftarrow (A) - ((H)(L)) - (CY)$	implicită + indirectă	toate	2/7 (7)
25	SBI <i>data8</i>	Subtract Immediate from A with Borrow	$(A) \leftarrow (A) - data8 - (CY)$	implicită + imediată	toate	2/7 (7)
26	INR <i>r</i>	INcRement register	$(r) \leftarrow (r) + 1$	la registru	toate	1/5 (4)
27	INR M	INcRement Memory	$((H)(L)) \leftarrow ((H)(L)) + 1$	indirectă	toate	3/10(10)
28	DCR <i>r</i>	DeCRement register	$(r) \leftarrow (r) - 1$	la registru	toate	1/5 (4)
29	DCR M	DeCRement Memory	$((H)(L)) \leftarrow ((H)(L)) - 1$	indirectă	toate	3/10(10)
30	INX <i>rp</i>	INcrement register pair	$(rp) \leftarrow (rp) + 1,$ cu $rp = B, D, H$ sau SP	la registru	-	1/5 (6)
31	DCX <i>rp</i>	DeCrement register pair	$(rp) \leftarrow (rp) - 1$	la registru	-	1/5 (6)
32	DAD <i>rp</i>	ADD <i>rp</i> to IIL	$(H)(L) \leftarrow (H)(L) + (r_H)(r_L)$	implicită + la registru	CY	3/10(10)
33	DAA	Decimal Adjust Accumulator	$(A_3 + A_0) > 9$ sau $(AC) = 1$ $\Rightarrow (A) \leftarrow (A) + 6;$ $(A_7 + A_4) > 9$ sau $(CY) = 1$ $\Rightarrow (A) \leftarrow (A) + 6 \times 2^4$	implicită	toate	1/4 (4)
Instrucțiuni logice						
34	ANA <i>r</i>	ANd register with A	$(A) \leftarrow (A) \wedge (r)$	implicită + la registru	Z, S, P; CY=0 AC=0	1/4 (4)

Tab.4.6 (cont.)

35	ANA M	ANd Memory with A	$(A) \leftarrow (A) \wedge ((H)(L))$	implicită + indirectă	Z, S, P; CY=0 AC=0	2/7 (7)
36	ANI data8	ANd Immediate with A	$(A) \leftarrow (A) \wedge data8$	implicită + imediată	Z, S, P; CY=0 AC=0	2/7 (7)
37	XRA r	eXclusive oR register with A	$(A) \leftarrow (A) \oplus (r)$	implicită + la registru	Z, S, P; CY=0 AC=0	1/4 (4)
38	XRA M	eXclusive oR Memory with A	$(A) \leftarrow (A) \oplus ((H)(L))$	implicită + indirectă	Z, S, P; CY=0 AC=0	2/7 (7)
39	XRI data8	eXclusive oR Immediate with A	$(A) \leftarrow (A) \oplus data8$	implicită + imediată	Z, S, P; CY=0 AC=0	2/7 (7)
40	ORI data8	OR Immediate with A	$(A) \leftarrow (A) \vee data8$	implicită + imediată	Z, S, P; CY=0 AC=0	2/7 (7)
41	ORA r	OR register with A	$(A) \leftarrow (A) \vee (r)$	implicită + la registru	Z, S, P; CY=0 AC=0	1/4 (4)
42	ORA M	OR Memory with A	$(A) \leftarrow (A) \vee ((H)(L))$	implicită + indirectă	Z, S, P; CY=0 AC=0	2/7 (7)
43	CMP r	CoMPare register with A	$(A) - (r)$	implicită + la registru	toate	1/4 (4)
44	CMP M	CoMPare Memory with A	$(A) - ((H)(L))$	implicită + indirectă	toate	2/7 (7)
45	CPI data8	ComPare Immediate with A	$(A) - data8$	implicită + imediată	toate	2/7 (7)
46	RLC	Rotate A Left with Carry	$(CY) \leftarrow (A_7) \rightarrow (A_0),$ $(A_{n+1}) \leftarrow (A_n), n=0 \div 6$	implicită	CY	1/4 (4)
47	RRC	Rotate A Right with Carry	$(CY) \leftarrow (A_0) \rightarrow (A_7),$ $(A_{n+1}) \rightarrow (A_n), n=0 \div 6$	implicită	CY	1/4 (4)
48	RAL	Rotate A Left through carry	$(A_0) \leftarrow (CY) \leftarrow (A_7),$ $(A_{n+1}) \leftarrow (A_n), n=0 \div 6$	implicită	CY	1/4 (4)
49	RAR	Rotate A Right through carry	$(A_0) \rightarrow (CY) \rightarrow (A_7),$ $(A_{n+1}) \rightarrow (A_n), n=0 \div 6$	implicită	CY	1/4 (4)
50	CMA	CoMplement Accumulator	$(A) \leftarrow \overline{(A)}$	implicită	-	1/4 (4)
51	STC	SeT Carry	$(CY) \leftarrow 1$	implicită	CY=1	1/4 (4)
52	CMC	CoMplement Carry	$(CY) \leftarrow \overline{(CY)}$	implicită	CY	1/4 (4)
Instrucțiuni de ramificare						
53	JMP addr	JuMP unconditional	$(PC) \leftarrow addr$	directă	-	3/10(10)
54	Jcc addr	Jump on condition cc	Dacă cc e adevărată, $(PC) \leftarrow addr;$ Altfel, $(PC) \leftarrow (PC) + 3$	directă	-	3/10(10)
						1/5 (7)

Tab.4.6 (cont.)

55	CALL addr	CALL unconditional	$(SP) \leftarrow (SP) - 1$ $((SP)) \leftarrow (PC_H)$ $(SP) \leftarrow (SP) - 1$ $((SP)) \leftarrow (PC_L)$ $(PC) \leftarrow addr$	directă + indirectă + implicită	-	5/17(18)
56	Ccc addr	Call on condition cc	Dacă cc e adevărată, $(SP) \leftarrow (SP) - 1$ $((SP)) \leftarrow (PC_H)$ $(SP) \leftarrow (SP) - 1$ $((SP)) \leftarrow (PC_L)$ $(PC) \leftarrow addr$; Altfel, $(PC) \leftarrow (PC) + 3$	directă + indirectă + implicită	-	5/17(18) 3/11 (9)
57	RET	RETurn	$(PC_L) \leftarrow ((SP))$ $(SP) \leftarrow (SP) + 1$ $(PC_H) \leftarrow ((SP) + 1)$ $(SP) \leftarrow (SP) + 1$	indirectă + implicită	-	3/10(10)
58	Rcc	Return on condition cc	Dacă cc e adevărată, $(PC_L) \leftarrow ((SP))$ $(SP) \leftarrow (SP) + 1$ $(PC_H) \leftarrow ((SP) + 1)$ $(SP) \leftarrow (SP) + 1$; Altfel, $(PC) \leftarrow (PC) + 1$	indirectă + implicită	-	3/11(12) 1/5 (6)
59	RST n	ReSTart n = 0, 1, 2, ..., 7	$(SP) \leftarrow (SP) - 1$ $((SP)) \leftarrow (PC_H)$ $(SP) \leftarrow (SP) - 1$ $((SP)) \leftarrow (PC_L)$ $(PC) \leftarrow n \times 8$	directă + indirectă + implicită	-	3/11(12)
60	PCHL	move HL to PC (jump HL indirect)	$(PC_H) \leftarrow (H)$ $(PC_L) \leftarrow (L)$	implicită	-	1/5 (6)
Instrucțiuni de lucru cu stiva, I/E și de comandă						
61	PUSH rp	PUSH register pair on stack rp = B, D sau H	$(SP) \leftarrow (SP) - 1$ $((SP)) \leftarrow (r_H)$ $(SP) \leftarrow (SP) - 1$ $((SP)) \leftarrow (r_L)$	la registru + indirectă + implicită	-	3/11(12)
62	PUSH PSW	PUSH accumulator and flags on stack; Flags: S Z 0 AC 0 P 0 CY	$(SP) \leftarrow (SP) - 1$ $((SP)) \leftarrow (A)$ $(SP) \leftarrow (SP) - 1$ $((SP)) \leftarrow (Flags)$	indirectă + implicită	-	3/11(12)
63	POP rp	POP register pair off stack rp = B, D sau H	$(r_L) \leftarrow ((SP))$ $(SP) \leftarrow (SP) + 1$ $(r_H) \leftarrow ((SP))$ $(SP) \leftarrow (SP) + 1$	la registru + indirectă + implicită	-	3/10(10)
64	POP PSW	POP accumulator and Flags off stack; Flags: S Z - AC - P - CY	$(Flags) \leftarrow ((SP))$ $(SP) \leftarrow (SP) + 1$ $(A) \leftarrow ((SP))$ $(SP) \leftarrow (SP) + 1$	indirectă + implicită	toate	3/10(10)
65	XTHL	eXchange stack Top with HL	$(L) \leftrightarrow ((SP))$ $(H) \leftrightarrow ((SP) + 1)$	implicită + indirectă	-	5/18(16)
66	SPHL	move HL to SP	$(SP) \leftarrow (HL)$	la registru	-	1/5 (6)

Tab.4.6 (cont.)

67	IN port	Input	$(A) \leftarrow (port)$	implicită + directă	-	3/10(10)
68	OUT port	OUTput	$(port) \leftarrow (A)$	implicită + directă	-	3/10(10)
69	EI	Enable Interrupts	$(INTE) \leftarrow 1$	implicită	-	1/4 (4)
70	DI	Disable Interrupts	$(INTE) \leftarrow 0$	implicită	-	1/4 (4)
71	HLT	HaLT	PC - blocat	-	-	1/7 (5)
72	NOP	No OPeration	$(PC) \leftarrow (PC) + 1$	-	-	1/4 (4)
Instrucțiuni specifice pentru 8085A						
73	SIM	Set Interrupt Mask and write serial bit	$SOD \leftarrow (A_7)$ și maschează liniile RST 7.5, 6.5 și 5.5	implicită	-	1/ (4)
74	RIM	Read serial bit and reset Interrupt Mask	$(A_7) \leftarrow SID$ și demaschează liniile RST 7.5, 6.5 și 5.5	implicită	-	1/ (4)

În Anexa I este prezentată lista completă a codurilor instrucțiunilor celor două microprocesoare Intel.

4.3.4. Programarea în limbaj de asamblare

Așa cum s-a arătat în §1.1.1.4, programarea în limbaj de asamblare se face în cadrul fixat de programul de asamblare automată, care are trei componente de bază:

- setul de instrucțiuni al microprocesorului (instrucțiuni executabile) (v.tab.4.6 și Anexa I);
- directivele de asamblare (pseudoinstrucțiuni);
- regulile sintactice pentru scrierea fișierului sursă.

Pentru microprocesoarele 8080/8085, atât Intel - ca firmă producătoare - cât și alte firme au realizat asambloare, macroasambloare și cross-asambloare absolute sau relocabile (ASM80 - Intel, M80 (Macro-80) - Microsoft, ASM, MAC, RMAC - Digital Research) care, cu mici diferențe, respectă aceleași reguli sintactice pentru fișierul sursă și utilizează un trunchi comun de directive de asamblare, așa cum a fost stabilit inițial în varianta elaborată de Intel.

4.3.4.1. Limbajul de asamblare Macro-80

În continuare se prezintă principalele directive de asamblare și reguli sintactice de scriere a fișierelor sursă, așa cum sunt definite de macroasamblorul M80 (Macro-80).

CSEG (Code SEGment), **DSEG** (Data SEGment), **ASEG** (Absolute SEGment). În general, se poate considera că un program executabil este format din instrucțiuni (cod) și date (constante și variabile). De regulă, zona de cod și datele constante ale programului nu se modifică pe parcursul execuției; în acest caz, ele pot fi rezidente într-o memorie de tip ROM (memorie program). Dimpotrivă, variabilele se modifică în timpul execuției programului și acest lucru trebuie să fie permis de circuitele de memorie în care se află alocate; de aici și obligativitatea amplasării lor numai în memorii de tip RAM (memorie de date). Drept urmare, încă din faza de scriere a fișierului sursă, programul trebuie divizat în două segmente: un *segment de cod*, care să grupeze toate instrucțiunile și datele constante ale programului, respectiv un *segment de date*, care să reunească toate variabilele definite în program; cele două segmente urmează să fie amplasate în zone diferite de memorie. Directiva CSEG marchează începutul unei secțiuni de program sursă care va face parte din segmentul de cod, iar directiva DSEG - începutul unei secțiuni care va face parte din segmentul de date. Directiva ASEG se poate folosi pentru a forța

amplasarea unei secvențe de program sau a unei zone de date la o adresă fixă, absolută, care nu mai este modificată la editarea legăturilor.

ORG *exp* - expresia *exp* precizează deplasamentul (offset-ul) adresei de memorie începând cu care vor fi amplasate instrucțiunile care urmează după ORG (ORiGine), față de începutul modulului relocabil curent (0000h). În cadrul operației de asamblare, contorul de program gestionat de asamblor va fi încărcat cu valoarea expresiei din câmpul operand, *exp*. Această pseudoinstrucțiune se folosește pentru a forța amplasarea unei secvențe de instrucțiuni la un anumit deplasament față de adresa de început a modulului de program curent.

const EQU *exp* (EQUate) - atribuie în mod *permanent* constantei cu numele simbolic *const* valoarea dată de expresia *exp* (o expresie numerică, logică sau un alt nume simbolic).

cstemp SET *exp* - atribuie în mod *temporar* constantei cu numele simbolic *cstemp* valoarea dată de expresia *exp*. Singura deosebire față de EQU este aceea că un simbol definit cu directiva SET se poate redefini ulterior, în cadrul aceluiași fișier sursă, ori de câte ori este nevoie. Asamblorul substituie numele simbolice *const* sau *cstemp* cu valorile atribuite cu ajutorul directivelor EQU și SET; de aceea, ele trebuie să preceadă orice utilizare a numelor simbolice respective.

[*etich:*] **DB** *lista* (Define Byte) - definește valoarea numelui simbolic *etich* (opțional) ca fiind adresa de început a unei zone de memorie alocată static, organizată pe octeți și inițializată cu valorile numerice ale elementelor din *listă*. Acestea pot fi valori numerice (0÷255), constante simbolice (definite anterior cu EQU sau SET), șiruri de caractere ASCII între ghilimele simple, expresii aritmetice și logice, toate separate prin virgulă. Directiva DB se utilizează de regulă pentru a defini date de 1 octet, constante sau variabile inițializate static; ultimul caz se utilizează numai dacă programul se încarcă și se execută într-o memorie de tip RAM.

[*etich:*] **DW** *lista* (Define Word) - la fel ca DB, dar zona de memorie alocată este organizată pe cuvinte de 2 octeți. În listă se află de obicei valori numerice (0÷65535) sau adrese simbolice (etichete) de instrucțiuni ori de date, definite în fișierul sursă; octetul mai puțin semnificativ se memorează la adresa mai mică.

[*etich:*] **DS** *exp* (Define Storage) - definește valoarea numelui simbolic *etich* (opțional) ca fiind adresa de început a unei zone de memorie, alocată static, de dimensiune egală cu valoarea numerică a expresiei *exp*, dar neinițializată. În câmpul operand, *exp*, se poate afla un număr, o constantă simbolică definită anterior (cu EQU sau SET) sau o expresie aritmetică.

PUBLIC *lista*, **EXTRN** *lista* - aceste directive se utilizează atunci când două sau mai multe fișiere sursă se referă la o aceeași adresă simbolică (etichetă), de cod sau de date. Unicitatea presupune că aceasta este definită o singură dată (prin *etich:*), numai într-unul din fișierele sursă; în acel fișier se utilizează directiva PUBLIC, pentru a o face astfel cunoscută în exterior, iar EXTRN trebuie utilizată în restul fișierelor care conțin referiri la acea adresă simbolică. Lista din câmpul de operand poate conține mai multe astfel se simboluri, separate prin virgulă.

O categorie distinctă o constituie directivele de *asamblare condiționată*. O secvență de asamblare condiționată are următoarea formă:

IF <i>arg</i>	<i>secvența 1</i>
[ELSE	<i>secvența 2</i>]
ENDIF	

Dacă argumentul este o expresie diferită de 0 - se assemblează *secvența 1*, altfel - se assemblează *secvența 2*. Directiva ELSE este opțională, dar ENDIF încheie întotdeauna secvența de asamblare condiționată. În locul directivei IF se pot folosi și alte variante (IF/IFT, IFE/IFF, IF1, IF2, IFDEF, IFNDEF etc.), care diferă numai prin modul de interpretare a argumentului *arg*.

END - indică sfârșitul fișierului sursă, fiind delimitatorul său final. Detectarea sa de către asamblor va determina sfârșitul unei faze (tregeri) a asamblării.

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

O linie sursă scrisă în limbajul de asamblare Macro-80 respectă regulile formale de sintaxă menționate în paragraful §1.1.1.4. Ea poate avea maximum 132 de caractere ASCII și conține, în principiu, patru câmpuri distincte: *eticheta*, *codul*, *operandul* și *comentariul*.

Câmpul etichetă - marchează numele simbolic al unei constante sau adrese, din care numai primele 6 caractere alfanumerice sunt semnificative. Primul caracter al etichetei trebuie să fie o literă sau unul din caracterele speciale: '\$', '.', '?', '@' sau '_'. După ultimul caracter al etichetei trebuie să urmeze caracterul ":" dacă ea marchează o adresă de memorie, având rol de *delimitator de câmp*. Utilizarea a două astfel de caractere consecutive ("::") este un alt mod, mai simplu, de a declara o adresă simbolică ca fiind de tip PUBLIC. Nu pot fi folosite ca etichete mnemonicele operațiilor sau pseudoinstrucțiunilor și nici numele registrelor interne ale procesorului.

Câmpul cod - admite numai mnemonicele operațiilor sau pseudoinstrucțiunile acceptate de asamblor (v.tab.4.6 și Anexa I). Pentru scrierea programului se pot folosi atât litere mari cât și litere mici, în orice context; asamblorul le tratează la fel.

Câmpul operand poate specifica: un registru, un registru pereche, o constantă numerică sau simbolică de 1 sau 2 octeți, o expresie aritmetică sau o etichetă din segmentul de cod ori de date. Când sunt doi operanzi, primul operand specifică destinația operației, iar al doilea specifică sursa; cei doi operanzi se separă prin virgulă. Operanzii pot fi exprimați în următoarele 9 moduri:

1. O *constantă zecimală*, urmată de litera 'd' (sau 'D') dacă baza de numerație implicită nu este baza 10. Deși este posibil să se fixeze orice bază implicită cu ajutorul directivei ".RADIX"; de cele mai multe ori se lucrează cu baza 10 ca bază implicită și, mai rar, baza 16. Exemplu:

Etichetă	Cod	Operand	Comentariu
aici:	MVI	A,58	; Baza de numerație implicită este 10. ; Inițializează acumulatorul cu constanta $58_{10} =$; $= 3 \times 16 + 10 = 3A_{16}$.
.RADIX 16			; Schimbă baza implicită de numerație (devine 16).
	LXI	SP,65535D	; Poziționează vârful stivei la adresa $65535_{10} = FFFF_{16}$.
.RADIX 10			; Baza 10 redevine baza de numerație implicită.

2. O *constantă hexazecimală*, care trebuie să înceapă cu o cifră (0÷9) și să se termine cu litera 'h' (sau 'H'), dacă baza de numerație implicită nu este baza 16, ca în exemplul următor:

aici:	MVI	A,3Ah	; Baza de numerație implicită este 10. ; Inițializează acumulatorul cu constanta $3A_{16} = 58_{10}$.
.RADIX 16			; Schimbă baza implicită de numerație (devine 16).
	LXI	SP,0FFFF	; Poziționează vârful stivei la adresa $FFFF_{16}$.
.RADIX 10			; Baza 10 redevine baza de numerație implicită.

3. O *constantă octală* care, dacă baza de numerație implicită nu este baza 8, trebuie să se termine cu litera 'q' ('Q').

	MVI	B,72Q	; Baza de numerație implicită este 10. ; Inițializează registrul B cu constanta $72_8 = 3A_{16} = 58_{10}$
--	-----	-------	---

4. O *constantă binară* care, dacă baza de numerație implicită nu este baza 2, trebuie să se termine cu litera 'b' ('B').

	ANI	00111010b	; Baza de numerație implicită este 10. ; Și logic între acumulator și constanta $00111010_2 =$; $= 72_8 = 3A_{16}$.
--	-----	-----------	---

5. Valoarea curentă a *contorului de program* (PC) sau *adresa instrucțiunii curente*, care se specifică prin caracterul '\$'.

salt:	JMP	\$+8	; Salt în program peste 8 locații de memorie, la ; adresa "salt+8".
-------	-----	------	--

6. O constantă ASCII, inclusă între ghilimele simple.

carac: MVI A, '@' ; Încarcă în acumulator codul ASCII al caracterului @
; (= 40h).

7. Numele unei *constante simbolice*, căreia asamblorul i-a atribuit o valoare numerică.

Port_D	EQU	40h	
	IN	Port_D	; Citește în registrul A octetul din portul de intrare cu ; adresa 40h.

8. Numele unei *adrese simbolice (etichete)* definite în program.

	LDA	contor	; Încarcă în acumulator octetul de la adresa "contor".
buc1a:	DEC	A	; Decrementează conținutul acumulatorului.
	JNZ	buc1a	; Dacă, după decrementare, A≠0, execută un salt la
	...		; instrucțiunea de la adresa "buc1a".
contor:	DS	1	; Rezervă un octet, la adresa "contor"

9. *Expresii aritmetice și logice* în care se folosesc toate tipurile de date descrise mai sus și care constituie operanzii expresiei. Operanzii sunt conectați cu ajutorul operatorilor aritmetici: +, -, *, /, MOD (modulo) sau logici: NOT, AND, OR, XOR, SHL, SHR (de deplasare la stânga sau la dreapta), precum și cu ajutorul parantezelor (stânga și dreapta). Lungimea operanzilor luați în considerare la evaluarea expresiilor de către asamblor este de 16 biți.

Operatorii aritmetici realizează, în ordinea enumerării lor de mai sus, *adunarea*, *scăderea*, *înmulțirea*, *împărțirea* întreagă, respectiv calculul *restului împărțirii* dintre 2 operanzi. Operatorii logici acționează la nivel de bit și produc *complementarea*, *produsul logic*, *suma logică* și respectiv *suma modulo 2* a argumentelor:

Operatorii SHL și SHR produc deplasarea liniară a primului operand spre stânga, respectiv spre dreapta, cu un număr de poziții egal cu valoarea celui de-al doilea operand. În partea opusă deplasării se introduce un număr de zerouri egal cu numărul de deplasări.

Ordinea în care sunt executate operațiile dintr-o expresie este următoarea:

- | | |
|---------------------------------|------------|
| 1. expresiile dintre paranteze, | 4. NOT |
| 2. *, /, MOD, SHL, SHR | 5. AND |
| 3. +, - | 6. OR, XOR |

Operatorii MOD, SHL, SHR, NOT, AND, OR și XOR trebuie separați de operanzi cu cel puțin un blank.

Câmpul comentariu trebuie precedat de un caracter ';', care îl separă de câmpul operand; de obicei acesta este precedat, pentru aliniere, de mai multe blankuri sau tabulatori (' ', TAB). Un comentariu se poate întinde pe mai multe rânduri, dar fiecare rând trebuie să înceapă cu un delimitator';'.

4.3.4.2. Tehnici de programare

În general, un program sursă în limbaj de asamblare se reprezintă ca o înlanțuire de linii sursă, care în formatul BNF (v. §1.1.1.4) arată astfel:

<PROGRAM> ::= IP <ȘIR DE LINII SURSĂ> SP.

în care IP și SP sunt delimitatorii de început, respectiv de sfârșit de program.

Liniile sursă vor fi despărțite prin delimitatori de linie, DL:

$$\langle \text{ȘIR DE LINII SURSĂ} \rangle ::= \langle \text{LINIE SURSĂ} \rangle | \langle \text{LINIE SURSĂ} \rangle \text{DL} \langle \text{ȘIR DE LINII SURSĂ} \rangle$$

Pentru a obține un program sursă eficient (memorie ocupată cât mai mică și durată de execuție cât mai redusă) este necesar să se optimizeze scrierea acestuia. Metodele care permit

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

atingerea acestui deziderat sunt denumite *tehnici de programare*. Deși tipul de procesor, prin resursele sale software marchează aceste tehnici, există și elemente de generalitate. Astfel, tehnicile de programare pentru microprocesoarele 8080/8085 se aplică în bună parte și la microprocesorul Z80. În continuare vor fi discutate câteva din aceste tehnici de programare; cititorii interesați pot aprofunda această problemă apelând la [27, 30, 39].

Implementarea operațiilor de control

O caracteristică importantă a programării microprocesoarelor este posibilitatea de luare a deciziilor. Programele sunt formate din mai multe secvențe liniare de instrucțiuni (ramuri), fiecare rezolvând o anumită sarcină. Aceste ramuri sunt executate într-o anumită ordine:

- impusă de programator, care le ordonează în fișierul sursă într-o anumită succesiune, sau
- determinată de microprocesor, la momentul execuției programului.

În acest mod se poate realiza controlul anumitor operații în funcție de mai multe condiții.

Implementarea operațiilor de control se bazează pe *testarea indicatorilor de condiție* ai microprocesorului. Cu ajutorul instrucțiunilor care țin cont de valoarea acestor indicatori se pot scrie structuri de control care să ia decizii după conținutul unor registre, valoarea unor biți sau valoarea unor cuvinte de 1 sau 2 octeți.

La scrierea structurilor de control trebuie rezolvate trei probleme: de *etichetare*, de *testare* și de *orientare* [5].

Etichetarea trebuie făcută pentru fiecare ramură din program care constituie ieșire pentru o instrucțiune de ramificare. Eticheta este de fapt adresa simbolică a începutului fiecărei ramuri de program.

Testarea condițiilor, pentru luarea deciziilor, se face cu ajutorul unor instrucțiuni care nu afectează datele programului.

Orientarea se face printr-una din mai multe ramuri posibile ale unui program, în funcție de rezultatul testului. Pentru realizarea orientării, contorul de program trebuie încărcat cu adresa ramurii selectate în urma testării.

Din perspectiva controlului execuției unui program, setul de instrucțiuni al microprocesoarelor 8080/8085 se împarte în două categorii:

1. instrucțiuni care nu pot modifica explicit contorul de program al microprocesorului (PC) și care se execută în ordinea scrierii lor în fișierul sursă.
2. instrucțiuni de testare și/sau ramificare (de orientare): necondiționate (JMP, CALL, RST) sau condiționate (Jcc, Ccc, Rcc) de valorile indicatorilor de condiții S, Z, P și CY (v.tab.4.6). Acestea sunt prevăzute cu posibilitatea de a schimba ordinea naturală de execuție a celorlalte instrucțiuni din program.

În aceste condiții, în limbajul de asamblare se pot implementa două tipuri de structuri logice de bază: *secvențe liniare*, formate numai cu instrucțiuni din prima categorie, respectiv *structuri simple de decizie*, folosind instrucțiuni din categoria a doua. Prin combinarea acestor două tipuri de instrucțiuni se pot construi toate structurile de control necesare în aplicații: de la structuri de decizie, din ce în ce mai complexe (GOTO, IF[ELSE], SWITCH), până la cicluri cu număr de pași cunoscut (FOR) sau necunoscut, cu test inițial (WHILE) sau final (DO WHILE) [3, 30, 38].

Așa cum reiese și din ultima clasificare, instrucțiunile de testare ale microprocesoarelor 8080/8085 (și Z80), sunt combinate cu cele de orientare, formând instrucțiuni de ramificare condiționată. Pentru exemplificare, se prezintă instrucțiunile de salt condiționat - Jcc *addr*:

JZ	<i>addr</i>	- salt dacă Z = 1 (dacă rezultatul ultimei operații aritmetice sau logice este 0);
JNZ	<i>addr</i>	- salt dacă Z = 0 (dacă rezultatul nu este 0);
JC	<i>addr</i>	- salt dacă CY = 1 (dacă a apărut transport/ împrumut);
JNC	<i>addr</i>	- salt dacă CY = 0 (dacă nu a apărut transport/ împrumut);

JPE *addr* - salt dacă $P = 1$ (dacă rezultatul are un număr par de biți "1");
 JPO *addr* - salt dacă $P = 0$ (dacă rezultatul are un număr impar de biți "1");
 JP *addr* - salt dacă $S = 0$ (dacă rezultatul este un număr pozitiv);
 JM *addr* - salt dacă $S = 1$ (dacă rezultatul este un număr negativ).

După cum se observă, testarea îndeplinirii condițiilor *cc* se referă direct la starea indicatorilor de condiție ai microprocesorului. Din acest motiv, luarea deciziilor în funcție de conținutul unor registre, de valoarea unor octeți sau cuvinte din memorie trebuie precedată de operații pregătitoare. Acestea au rolul de a transfera starea la nivelul indicatorilor de condiție ai microprocesorului. Aceste operații se pot efectua cu ajutorul unor instrucțiuni aritmetice sau logice, așa cum sunt primele două instrucțiuni din exemplul care urmează.

Exemplu: În funcție de valoarea octetului conținut de registrul H: negativ, 0 sau pozitiv, să se incrementeze registrul E, să se decrementeze registrul D, respectiv să se incrementeze registrul C. În fig.4.24 este prezentată organigrama, cu specificarea etichetelor care marchează începutul ramurilor: *start*, *test1*, *test2* și *cont*, așa cum apar în programul listat în continuare.

start:	XRA	A	; Forțază 0 în A ($x \oplus x = 0$).
	ADD	H	; Adună A cu conținutul lui H.
	JM	test2	; Salt la test2 dacă $(A) = (H) < 0$
	JZ	test1	; Salt la test1 dacă $(A) = (H) = 0$
	INR	C	; Tratează cazul $(A) = (H) > 0$.
	JMP	cont	; Continuă de la adresa cont.
test1:	DCR	D	; Tratează cazul $(A) = (H) = 0$
	JMP	cont	
test2:	INR	E	; Tratează cazul $(A) = (H) < 0$
cont:			

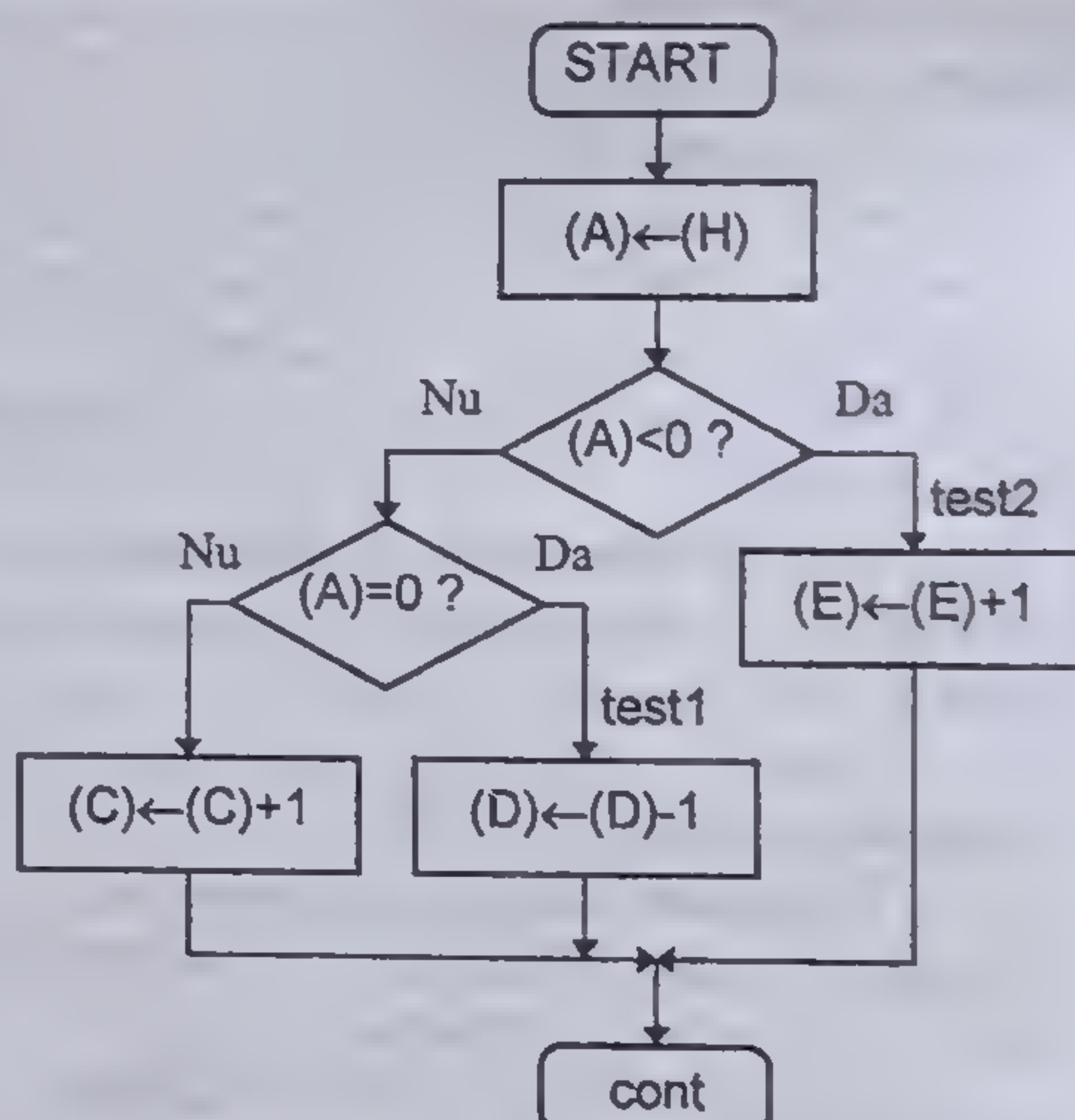


Fig.4.24. Testarea semnului octetului din H

Testarea valorii anumitor biți dintr-un

octet este necesară în numeroase situații. În acest scop, se poate proceda după cum urmează:

1. octetul care conține biții ce trebuie testați se transferă mai întâi în acumulator;
2. se rețin apoi numai biții care interesează, prin *mascarea* celorlalți, adică prin forțarea lor pe "0" cu ajutorul unui ȘI logic;
3. cu ajutorul instrucțiunilor de comparare, scădere sau sumă modulo 2 se poziționează indicatorii de condiție;
4. în final se utilizează instrucțiuni de ramificare, condiționate de starea unui anumit indicator.

Exemplu: Să se stabilească dacă toți biții de pe rangurile impare (D_7, D_5, \dots, D_1) ai octetului din locația de memorie de la adresa *adr* sunt la valoarea "1". În caz afirmativ, programul se ramifică la adresa *adr1*.

LDA	adr	; Încarcă în A octetul din memorie, de la adresa "adr".
ANI	10101010b	; Forțază pe "0" biții de pe rangurile pare.
CPI	10101010b	; Testează dacă biții de pe rangurile impare sunt "1".
JZ	adr1	; Dacă da (rezultatul este 0), se sare la adresa "adr1"
...		; altfel, continuă.

Un alt procedeu, utilizat de obicei pentru testarea succesivă a unor biți din același octet, constă din încărcarea octetului respectiv în acumulator, rotirea lui la stânga sau la dreapta de un anumit număr de ori, până la aducerea bitului dorit în CY, urmată apoi de testarea acestui indicator.

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

Exemplu: Să se testeze, pe rând, valorile biților de pe rangurile 0 și 2 din octetul citit din portul de intrare Port_C, astfel: dacă bitul de pe rangul 0 are valoarea "1", să se efectueze un salt la adresa adr0; aici, dacă bitul de pe rangul 2 are valoarea "0", să se sară la adresa adr2.

	IN	Port_C	; Citește în acumulator octetul din portul de intrare Port_C (valport).
	RRC		; Rotește octetul din A cu o poziție, astfel încât valport[0]→CY.
	JC	adr0	; Dacă CY=valport[0] este "1", sare la adresa "adr0";
	...		; altfel, continuă.
adr0:	RRC		; Rotește octetul din A cu o poziție, astfel încât valport[1]→CY.
	RRC		; Rotește octetul din acumulator cu o poziție, astfel încât valport[2]→CY.
	JNC	adr2	; Dacă CY=valport[2] este "0", sare la adresa "adr2";
	...		; altfel, continuă.
adr2:	...		; Aici se ajunge dacă valport[0]=1 și valport[2]=0.

Uneori este necesară memorarea temporară a indicatorilor de condiție, pentru a preîntâmpina alterarea lor în diverse secvențe de program, în special în cazul rutinelor de tratare a întreruperilor. Operațiile de memorare și restaurare a indicatorilor se realizează cu ajutorul instrucțiunilor PUSH PSW și POP PSW, folosind stiva.

Programarea cu macroinstrucțiuni

Programarea cu macroinstrucțiuni face parte din *tehnicele de optimizare* a programelor. La realizarea programelor sursă există operații de interes local, care se cer executate de mai multe ori în același program, dar cu alți parametri. Pentru a nu scrie de mai multe ori aceeași secvență de instrucțiuni, macroasamblearele permit definirea unui subprogram ca o *macroinstrucțiune*. O astfel de entitate este un grup de instrucțiuni care rezolvă o anumită operație, precedat de o pseudoinstrucțiune MACRO și terminată cu pseudoinstrucțiunea ENDM. Folosirea macroinstrucțiunilor la programarea în limbajul de asamblare duce la o scurtare și la o modularizare a fișierului sursă.

Programarea cu macroinstrucțiuni implică două faze distincte și anume: *definirea* și *referirea* macroinstrucțiunii.

Definirea macroinstrucțiunii în limbajul Macro-80 are următoarea sintaxă:

nume:	MACRO	arg1, arg2, ...	; Începutul macroinstrucțiunii.
.....			; Corpul macroinstrucțiunii.
	ENDM		; Sfârșitul macroinstrucțiunii.

Odată definită, o macroinstrucțiune se poate referi ori de câte ori este nevoie, prin simpla inserare în program a numelui și a parametrilor acesteia.

Exemplu: Să se definească o macroinstrucțiune (ADUNA) care să adune conținutul a două locații de memorie de 1 octet, iar rezultatul să se regăsească în acumulator. Folosind această macroinstrucțiune, să se adune conținutul locației a1 cu octetul de la adresa a2, rezultatul să se depună la a3, apoi să se adune octeții de la adresele b1 și b2, iar rezultatul să se memoreze la adresa b3.

ADUNA:MACRO	par1, par2	; Definirea macroinstrucțiunii ADUNA, cu 2 parametri formali.
LDA	par1	; Încărcarea în acumulator a primului termen.
MOV	B,A	; Primul termen este memorat în B.
LDA	par2	; Se încarcă cel de-al doilea termen (de la adresa par2).
ADD	B	; Se adună cu primul termen; rezultatul se obține în A.
ENDM		; Sfârșitul macroinstrucțiunii ADUNA.
...		
ADUNA	a1,a2	; Calculează (a1)+(a2) printr-o referire a macroinstrucțiunii ADUNA.
STA	a3	; Depune rezultatul în memorie, la adresa a3.
ADUNA	b2,b1	; Calculează (b2)+(b1).
STA	b3	; Depune rezultatul în memorie, la adresa b3.

La asamblarea programului, macroasamblorul va *expandă* toate referirile la macroinstrucțiune, adică va înlocui linia de referire cu corpul macroinstrucțiunii, în care argumentele formale vor fi substituite cu parametrii efectivi (cei utilizați la referirea macroinstrucțiunii).

Prin utilizarea macroinstrucțiunilor se economisește timpul programatorului, prin micșorarea timpului de scriere a programului sursă. Un al doilea avantaj îl constituie reducerea timpului de modificare și de verificare a unui program. Rezultă astfel o optimizare a codului sursă.

Însă referirile succesive ale unei macroinstrucțiuni conduc la repetarea, în programul generat, a aceleiași secvențe de instrucțiuni și la creșterea exagerată a dimensiunii codului executabil. Acest dezavantaj poate fi evitat prin utilizarea *subrutinelor*, așa cum se va arăta în continuare.

Programarea cu subrutine

Subrutina este o secvență de program care poate fi *apelată* ori de câte ori este nevoie, dar pentru care se alocă memorie numai o singură dată. Întrucât implementează o anumită funcție, subrutina poate fi privită ca un program independent. Câteva exemple de operații generale, care se pretează la programarea lor ca subrutine, sunt: operațiile I/E, conversiile de date dintr-un format în altul, operațiile aritmetice în virgulă fixă sau în virgulă mobilă, manipularea de liste etc.

Limbajul Macro-80 nu are o directivă de *definire* a subrutinelor, așa cum are pentru macroinstrucțiuni. Pentru ca o succesiune de instrucțiuni să constituie o subrutină, este suficient ca aceasta să aibă un nume, reprezentat de o etichetă a unei instrucțiuni din secvență (de obicei a primei instrucțiuni). În plus, execuția subrutinei trebuie să se termine cu o instrucțiune de revenire, de tipul RET (sau RCC), care nu întotdeauna este ultima din secvență. O altă particularitate este aceea că pot să existe mai multe puncte de intrare într-o subrutină, fiecare marcat de o etichetă, tot așa cum pot să existe mai multe puncte de ieșire din subrutină, fiecare printr-o altă instrucțiune de revenire.

Programul principal trebuie să conțină, în punctele în care este necesară apelarea subrutinei, o instrucțiune de ramificare de tip CALL (sau CCC), al cărei operand trebuie să fie numele subrutinei.

În general, la utilizarea subrutinelor apar următoarele probleme:

- realizarea legăturii dintre programul apelant și subrutină, cu asigurarea revenirii din subrutină;
- folosirea registrelor interne de către cele două unități de program;
- transferul de date între programul apelant și subrutină.

Pentru a scrie corect un program cu subrutine este necesar să se asigure legătura dintre subrutine și programul apelant. Dacă pentru apel se folosește instrucțiunea CALL și revenirea se face cu RET, organizarea legăturii cu programul principal corespunde schemei din fig.4.25.

Perechea de instrucțiuni CALL și RET asigură salvarea, respectiv refacerea automată a număratorului de program al microprocesorului (PC), care conține în orice moment adresa instrucțiunii ce urmează să se execute.

Pentru creșterea lizibilității codului sursă, fiecare subrutină trebuie documentată la definire, precizându-se:

- numele și funcția îndeplinită;
- modul în care sunt primiți parametrii de intrare;
- modul în care rezultatele sunt returnate programului apelant;
- registrele modificate în cadrul subrutinei.

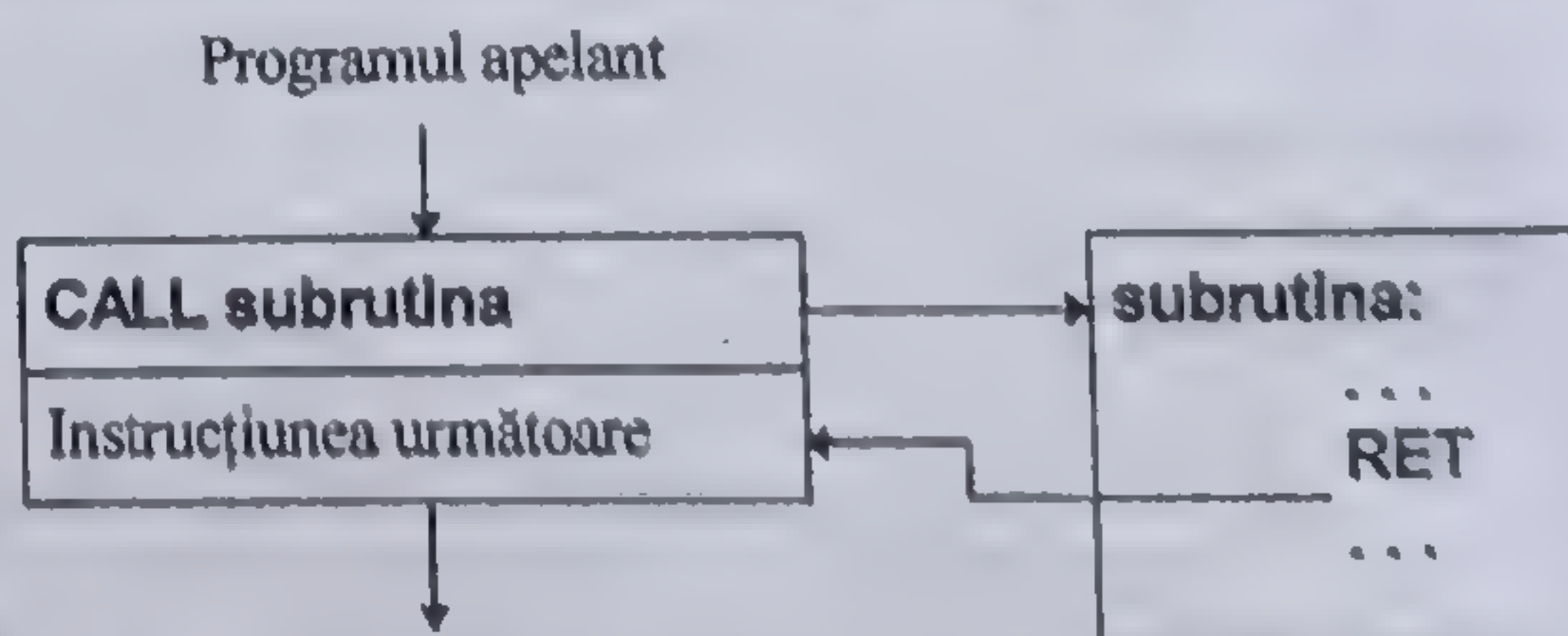


Fig.4.25. Apelul unei subrutine dintr-un program

La asamblarea programului, macroasamblorul va *expanda* toate referirile la macroinstrucțiune, adică va înlocui linia de referire cu corpul macroinstrucțiunii, în care argumentele formale vor fi substituite cu parametri efectivi (cei utilizați la referirea macroinstrucțiunii).

Prin utilizarea macroinstrucțiunilor se economisește timpul programatorului, prin micșorarea timpului de scriere a programului sursă. Un al doilea avantaj îl constituie reducerea timpului de modificare și de verificare a unui program. Rezultă astfel o optimizare a codului sursă.

Însă referirile succesive ale unei macroinstrucțiuni conduc la repetarea, în programul generat, a aceleiași secvențe de instrucțiuni și la creșterea exagerată a dimensiunii codului executabil. Acest dezavantaj poate fi evitat prin utilizarea *subrutinelor*, așa cum se va arăta în continuare.

Programarea cu subrutine

Subrutina este o secvență de program care poate fi *apelată* ori de câte ori este nevoie, dar pentru care se alocă memorie numai o singură dată. Întrucât implementează o anumită funcție, subrutina poate fi privită ca un program independent. Câteva exemple de operații generale, care se pretează la programarea lor ca subrutine, sunt: operațiile I/E, conversiile de date dintr-un format în altul, operațiile aritmetice în virgulă fixă sau în virgulă mobilă, manipularea de liste etc.

Limbajul Macro-80 nu are o directivă de *definire* a subrutinelor, așa cum are pentru macroinstrucțiuni. Pentru ca o succesiune de instrucțiuni să constituie o subrutină, este suficient ca aceasta să aibă un nume, reprezentat de o etichetă a unei instrucțiuni din secvență (de obicei a primei instrucțiuni). În plus, execuția subrutinei trebuie să se termine cu o instrucțiune de revenire, de tipul RET (sau Rcc), care nu întotdeauna este ultima din secvență. O altă particularitate este aceea că pot să existe mai multe puncte de intrare într-o subrutină, fiecare marcat de o etichetă, tot așa cum pot să existe mai multe puncte de ieșire din subrutină, fiecare printr-o altă instrucțiune de revenire.

Programul principal trebuie să conțină, în punctele în care este necesară apelarea subrutinei, o instrucțiune de ramificare de tip CALL (sau Ccc), al cărei operand trebuie să fie numele subrutinei.

În general, la utilizarea subrutinelor apar următoarele probleme:

- realizarea legăturii dintre programul apelant și subrutină, cu asigurarea revenirii din subrutină;
- folosirea registrelor interne de către cele două unități de program;
- transferul de date între programul apelant și subrutină.

Pentru a scrie corect un program cu subrutine este necesar să se asigure legătura dintre subrutine și programul apelant. Dacă pentru apel se folosește instrucțiunea CALL și revenirea se face cu RET, organizarea legăturii cu programul principal corespunde schemei din fig.4.25.

Perechea de instrucțiuni CALL și RET asigură salvarea, respectiv refacerea automată a număratorului de program al microprocesorului (PC), care conține în orice moment adresa instrucțiunii ce urmează să se execute.

Pentru creșterea lizibilității codului sursă, fiecare subrutină trebuie documentată la definire, precizându-se:

- numele și funcția îndeplinită;
- modul în care sunt primiți parametrii de intrare;
- modul în care rezultatele sunt returnate programului apelant;
- registrele modificate în cadrul subrutinei.

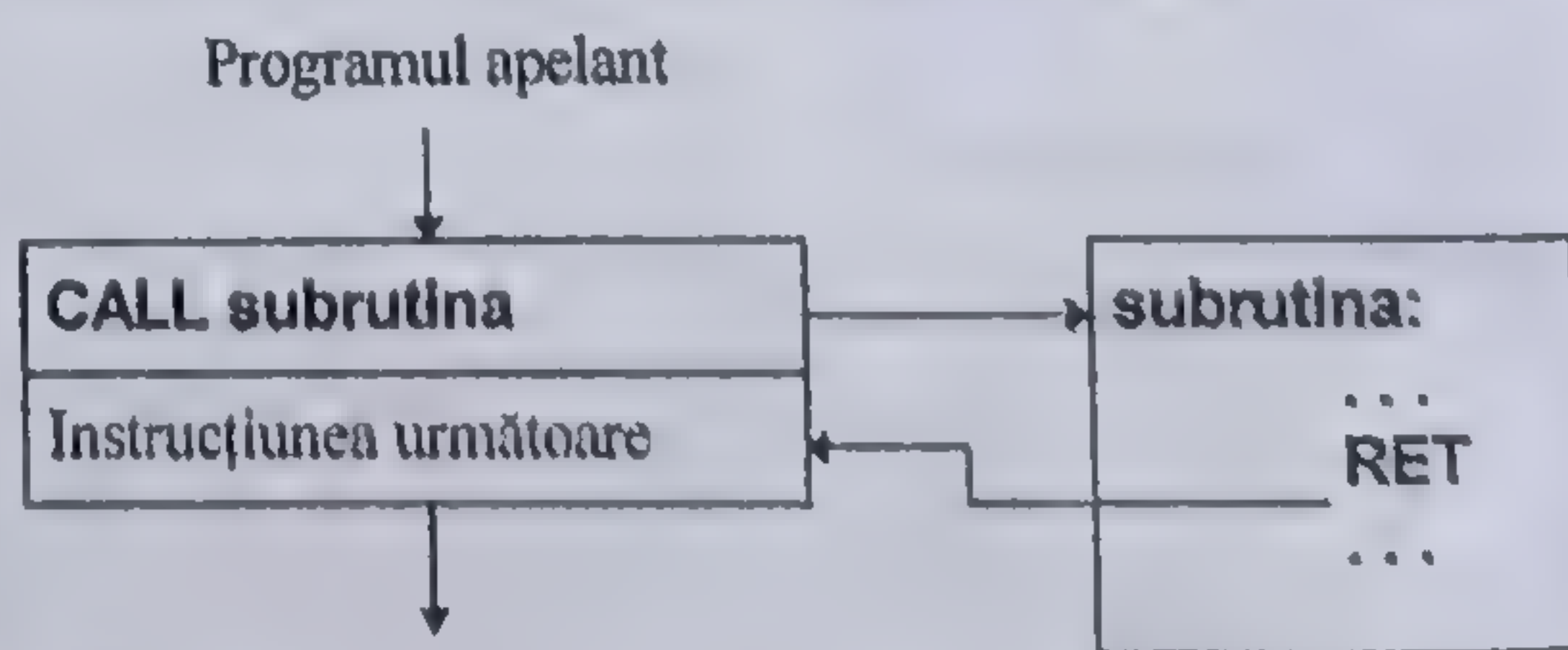


Fig.4.25. Apelul unei subrutine dintr-un program

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

Exemplu: Să se scrie o subrutină care transferă un număr de octeți (nrb) de la adresa sursă la adresa dest.

Se folosesc registrele pereche H și D pentru adresarea indirectă a locațiilor din zonele de memorie sursă, respectiv destinație, iar registrul C va conținea numărul de octeți rămași de transferat. Subrutina se va numi memcpy (copie o zonă de memorie în altă zonă).

; Programul apelant

```

...
LXI    H,sursa      ; Se încarcă în registrul pereche H adresa zonei sursă.
LXI    D,dest       ; Se inițializează DE cu adresa de început a zonei destinație.
MVI    C,nrb        ; Se încarcă în C numărul inițial de octeți de transferat.
CALL   memcpy       ; Se apelează rutina de transfer.
...

```

; MEMCPY - rutina de copiere a unei zone de memorie în altă zonă

Intrare: HL = adresa de început a zonei sursă
DE = adresa de început a zonei destinație
C = numărul de octeți de transferat.

Ieșire: zona de memorie destinație are același conținut cu zona de memorie sursă.
Distruge: conținutul registrelor A, C, DE, HL, indicatorii de condiție.

```

memcpy:
MOV     A,M          ; Punctul de intrare în subrutină, identic cu numele subrutinei.
STAX    D            ; Citește în A octetul din zona sursă, de la adresa din HL.
INX     H            ; Inscrie octetul din A în zona destinație, la adresa din DE.
INX     D            ; Incrementează registrul pointer pentru zona sursă (HL)
INX     D            ; Incrementează registrul pointer în zona destinație (DE).
DCR     C            ; Decrementează numărul de octeți rămași de transferat.
JNZ     memcpy       ; Dacă mai sunt octeți, sare la începutul subrutinei;
RET                      ; altfel, revine în programul apelant

```

Dacă în corpul subrutinei se modifică conținutul unor registre interne sau indicatorii de condiție, iar programul apelant dorește conservarea lor, este necesar ca fie programul apelant, fie subrutina apelată să prevadă salvarea, respectiv refacerea lor din stivă. În exemplul de mai jos, subrutina salvează și reface toate registrele prin intermediul stivei, ceea ce face ca programul apelant să poată conta pe aceleași valori ale acestora, la revenirea din subrutină.

; Programul apelant;

```

...
CALL   subrutina     ; Starea registrelor dinainte de apelul subrutinei.
...

```

; SUBRUTINA -

Intrare: ...

Ieșire: ...

Distruge: Nimic

subrutina:

```

PUSH    PSW          ; Salvează pe stivă acumulatorul și indicatorii de condiție.
PUSH    B            ; Salvează pe stivă conținutul registrului pereche B (B și C).
PUSH    D            ; Salvează pe stivă D (D și E).
PUSH    H            ; Salvează pe stivă H (H și L).
...
POP     H            ; corpur subrutinei, în care se modifică registrele și indicatorii de condiție.
POP     D            ; Reface de pe stivă, în H și L, conținutul salvat la intrarea în subrutină.
POP     B            ; Reface de pe stivă B.
POP     PSW          ; Reface de pe stivă acumulatorul și indicatorii de condiție.
RET                      ; Revine din subrutină, cu conținutul registrelor neschimbat.

```


O subrutină are toate atributele unui program: primește date de intrare de la un alt program (apelant), le prelucerează, iar la revenire îi furnizează acestuia rezultatele obținute. Cel mai simplu și mai eficient mod de a transfera date între cele două unități de program se realizează *prin intermediul registrelor interne* ale microprocesorului, așa cum se poate vedea și în exemplul cu rutina de copiere a unei zone de memorie. Parametrii efectivi de intrare ai subrutinei (adresele de început ale zonelor sursă, respectiv destinație și numărul de octeți de transferat) sunt încărcați în registrele interne ale microprocesorului (HL, DE, C), care corespund parametrilor formali ai acesteia. În mod similar, subrutina ar putea transmite rezultatele către programul apelant.

Atunci când numărul și/sau dimensiunea parametrilor este mai mare decât numărul de registre ale microprocesorului, transferul parametrilor și al rezultatelor se poate face prin intermediul unor *zone de memorie*. Astfel, datele pot fi memorate la adrese succesive de memorie, iar adresa zonei poate fi comunicată subrutinei prin intermediul unui registru pereche. Cea mai simplă metodă de transfer al datelor prin intermediul memoriei utilizează *stiva*; adresa de început a stivei este conținută implicit de registrul SP, accesibil atât programului apelant cât și subrutinei apelate.

Programarea cu subrutine este atât o tehnică de optimizare a programelor sursă, prin modularizarea acestora, cât și a codului executabil generat, contribuind la utilizarea eficientă a memoriei. Dintre dezavantaje se pot menționa: 1) mărirea timpului de execuție a programului, fiecare apel de subrutină necesitând execuția suplimentară a cel puțin unei instrucțiuni CALL și a unei instrucțiuni RET; 2) necesită existența stivei, deci impune prezența în sistem a unei memorii de tip RAM.

4.4. Sisteme pentru tratarea cererilor multiple de întrerupere

După cum a mai fost menționat, funcționarea în timp real a sistemelor cu microprocesoare este posibilă prin folosirea regimului de întreruperi. Dar, microprocesoarele prezintă dezavantajul unui număr de linii de întrerupere mult mai redus decât numărul real al solicitărilor. Microprocesorul 8080A posedă, după cum s-a văzut, o singură linie pentru solicitarea de întrerupere, iar 8085A un număr de 5 linii, din care 4 sunt cu adresă fixă de tratare. Din acest motiv, trebuie realizate sisteme care să permită microprocesorului tratarea solicitărilor multiple de întrerupere. În general, astfel de solicitări apar din partea unor dispozitive I/E, care necesită la un moment dat atenția microprocesorului în vederea realizării unei operații de intrare-ieșire (v. §2.3.2). Pentru recunoașterea dispozitivului ce a generat întreruperea se poate folosi atât tehnica de interogare (polling), cât și cea a întreruperilor vectorizate (vectoring). În ambele cazuri trebuie realizată arbitrarea solicitărilor simultane pe baza unui sistem de priorități în servire.

4.4.1. Sisteme de tratare a întreruperilor bazate pe tehnica de interogare

În acest caz dispozitivul I/E determină, prin activarea liniei de întrerupere a microprocesorului, declanșarea unei proceduri de interogare. Un exemplu tipic este cel din fig.4.26, în care se utilizează facilitatea dispozitivului 8228 de a forța pe magistrala de date codul instrucțiunii RST 7, atunci când linia $\overline{\text{INTA}}$ se conectează la +12V (v. §4.1.4.2).

O altă variantă folosește un port distinct (notat cu PINT în fig.4.26), care asigură forțarea pe magistrala de date a unei instrucțiuni RST n , ca răspuns la acceptarea întreruperii; această soluție necesită însă hardware suplimentar.

Presupunem că există 3 dispozitive I/E care pot solicita întrerupere. Cele trei semnale de întrerupere sunt combinate, cu ajutorul unei porți SAU, într-un singur semnal, care comandă intrarea de întrerupere a microprocesorului (INT). În același timp, cele trei linii sunt conectate și la liniile mai puțin semnificative ale unui port de intrare (PORT STARE INT), care poate fi citit de

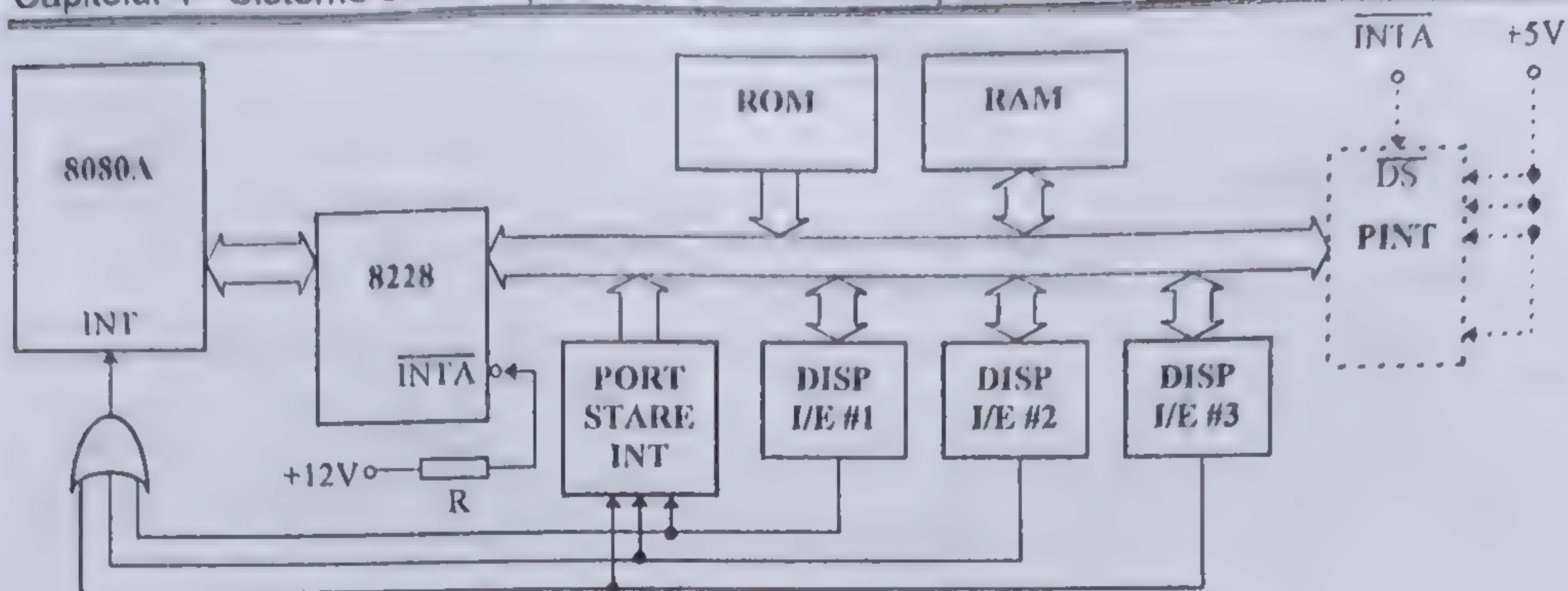


Fig.4.26. Sisteme de tratare a întreruperilor prin interogare

către microprocesor. Cuvântul citit reflectă, în cei mai puțin semnificativi 3 biți, existența solicitărilor de întrerupere ale celor trei dispozitive. După cum s-a văzut în §2.3.2, pentru identificarea sursei de întrerupere și pentru rezolvarea priorităților este necesar un program de interogare, în conformitate cu organigrama din fig.4.27.

În cazul în care unul sau mai multe dispozitive I/E solicită atenția microprocesorului, acesta va răspunde, în ciclul de întrerupere, prin activarea bitului $D_0 = \text{INTA}$ din cuvântul de stare. În acest moment, 8228 va forța pe magistrala de date codul FFh, corespunzător instrucțiunii RST 7. Microprocesorul îl citește, îl decodifică, apoi trece la execuția lui. Astfel, după salvarea registrului PC în stivă, microprocesorul va realiza un salt la adresa $8 \times 7 = 56_{10} = 0038_{16}$. La această adresă, notată cu poll în exemplul dat, trebuie să se afle programul de interogare, care testează biții corespunzători din portul de stare (PSTARE). Ordinea de interogare stabilește prioritatea de tratare. În momentul în care este detectată o cerere activă, se trece la execuția subrutinei de tratare corespunzătoare.

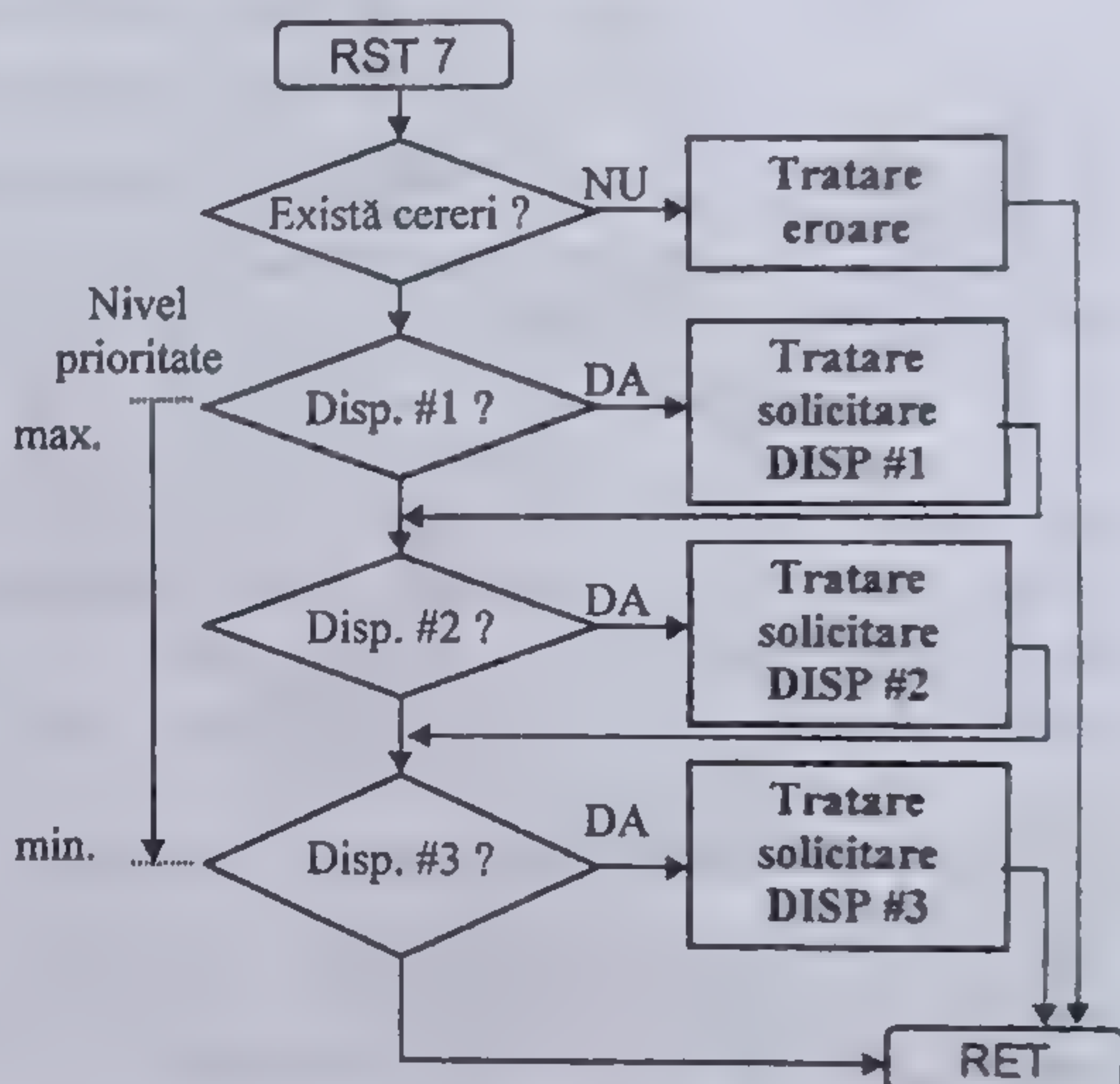


Fig.4.27. Programul de interogare

ORG 0038h	; Amplasează rutina poll la adresa 0038h
poll: PUSH PSW	; Salvează Acc și flag-urile.
PUSH B	; Salvarea registrelor de lucru.
PUSH D	
PUSH H	
IN PSTARE	; Citește starea dispozitivelor I/E
ANI 00000111B	
JZ eroare	; Tratează eroarea
RRC	; Este DISP #1 ?
CC trat1	; Dacă da, apelează rutina de tratare.
RRC	; Este DISP #2 ?
CC trat2	; Dacă da, apelează rutina de tratare.
RRC	; Este DISP #3 ?

CC	trat3	: Dacă da, apelează rutina de tratare.
POP	H	: Reface de pe stivă conținutul registrelor.
POP	D	
POP	B	
POP	PSW	:
EI		: Revalidează întreruperile pe linia INT (INTE←1).
RET		: Revine în programul întrerupt.

Schema funcționează corect numai dacă dispozitivul în curs de tratare își anulează cererea la începerea servirii, pentru a nu bloca linia INT în "1" logic. Numită și *achitare a cererii de întrerupere*, această operație poate fi efectuată implicit de către dispozitivul I/E (în momentul dialogului cu microprocesorul) sau trebuie prevăzută o comandă explicită în acest scop. Ca urmare, o rutină de tratare pentru exemplul de mai sus trebuie să aibă următoarea structură:

trat _i :		: i = 1, 2 sau 3
	PUSH PSW	
	MVI A, achit	: A = octetul de comandă pentru achitarea cererii de întrerupere
	OUT DISPi	: Achitare explicită a solicitării de la DISP i
	...	: Tratarea propriu-zisă a cererii de întrerupere.
	POP PSW	
	RET	: Revine în rutina de interogare, poll.

De asemenea, la revenirea din poll trebuie reactivate întreruperile printr-o instrucțiune EI, pentru ca o nouă solicitare să poată să fie luată în considerare.

Se păstrează în acest caz o parte din dezavantajele tehnicii de interogare: prioritate fixă în servire și un timp de răspuns relativ ridicat. Tratarea unei solicitări nu poate fi întreruptă de nici o altă cerere, chiar de la un dispozitiv mai prioritar. În schimb, folosirea întreruperilor permite procesorului să lanseze procedura de interogare numai în momentul apariției unei solicitări externe. La un număr mare de surse de întrerupere, de ordinul zecilor, timpul de răspuns al sistemului devine inacceptabil de mare pentru o funcționare în timp real.

4.4.2. Sisteme de întreruperi vectorizate

Tehnica întreruperilor vectorizate permite tratarea eficientă a unui număr mare de solicitări, la sistemele cu microprocesoare Intel fiind realizate dispozitive specializate pentru construirea de sisteme de întreruperi (SINT) multiple, performante.

Un prim dispozitiv este circuitul I8214, care este un controler de întreruperi pentru 8 cereri. Circuitul asigură rezolvarea priorităților în cazul solicitărilor simultane prin compararea acestora cu conținutul unui registru, controlat prin program. Cererii de prioritate maximă la momentul respectiv îi este asociat un vector, sub forma unei instrucțiuni RST *n*, care va fi transmis microprocesorului ca răspuns la semnalul $\overline{\text{INTA}}$. Avantajul acestui controler constă în eliminarea procedurii de interogare software a dispozitivelor solicitante. Dezavantajul constă în faptul că nu poate genera singur codul instrucțiunii RST *n*, fiind utilizat în acest scop un port 8212 [5, 24]. De asemenea, realizarea unui SINT cu mai mult de 8 linii devine relativ complicată, iar folosirea instrucțiunilor RST *n* reduce flexibilitatea sistemului.

4.4.2.1. Controlerole de întreruperi 8259/8259A

O altă generație de dispozitive, care elimină dezavantajele precedentelor, sunt *controlerolele programabile de întreruperi I8259/8259A* (PIC - Programmable Interrupt Controller). Acestea sunt de fapt SINT-uri realizate sub formă de circuite integrate și care rezolvă principalele sarcini ale unui astfel de sistem:

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

- acceptă cereri de la mai multe surse de întrerupere;
- determină cererea de prioritate maximă;
- întrerup activitatea unității centrale la apariția a cel puțin unei solicitări;
- informează unitatea centrală, prin instrucțiuni de tip CALL *addr* (la 8080/8085) sau INT *n* (la 8086), asupra adresei de start a rutinei care deservește dispozitivul solicitant;
- se pot utiliza pentru realizarea de sisteme cu până la 64 de niveluri de întrerupere, prin conectarea lor în cascadă.

Structural, cele două controlere sunt asemănătoare: 8259A este o versiune îmbunătățită a circuitului 8259, fiind capabil să lucreze cu procesoare Intel atât de 8, cât și de 16 biți.

În fig.4.28 se prezintă schema bloc a celor două dispozitive. Ambele circuite sunt împachetate în capsule cu 28 de pini, fiind alimentate la o singură sursă de +5V [42].

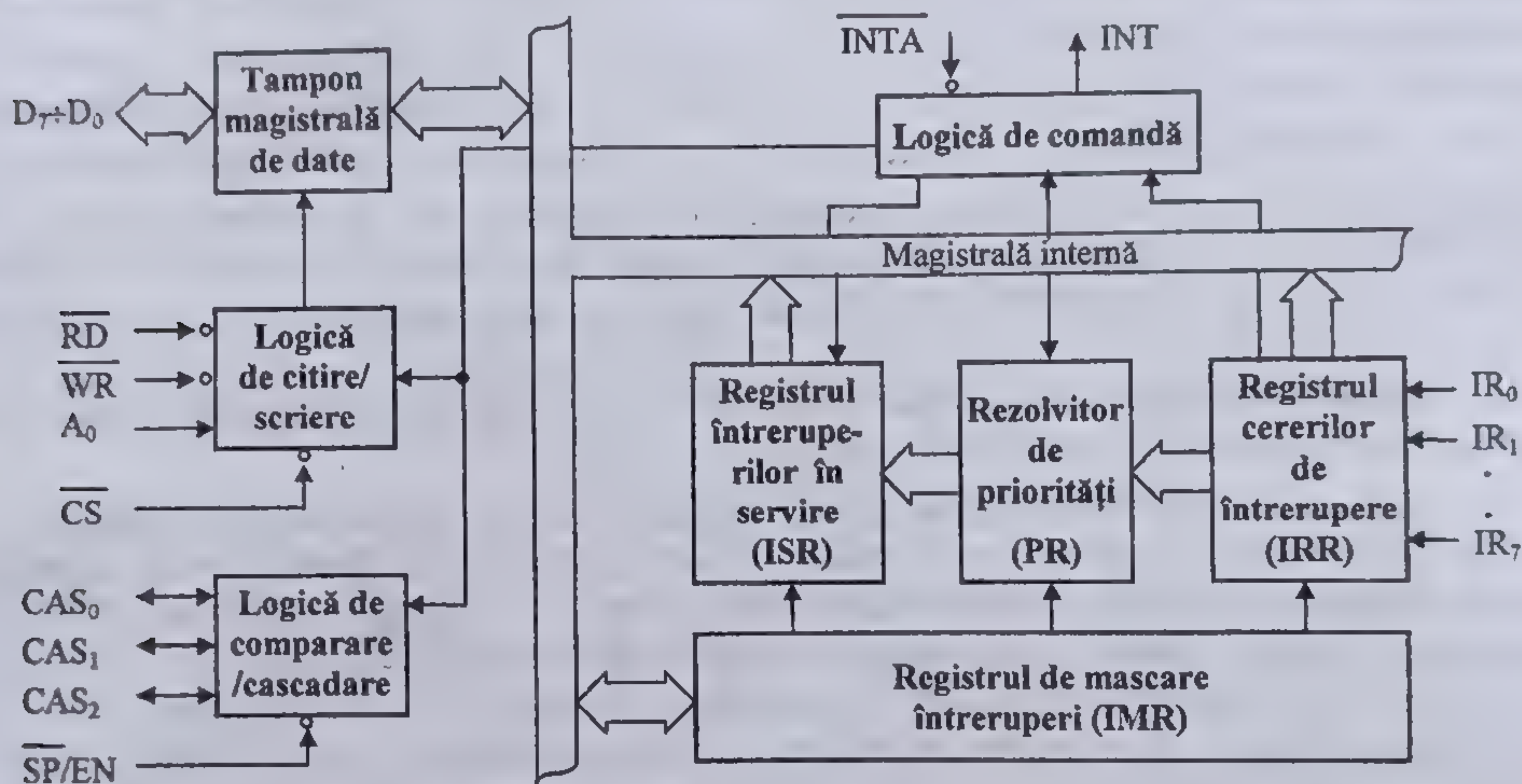


Fig.4.28. Schema bloc a circuitelor PIC 8259/8259A

Cererile de întrerupere se aplică pe liniile $IR_0 \div IR_7$. Registrul cererilor de întrerupere, IRR (Interrupt Request Register), memorează aceste cereri și le transmite blocului de rezolvare a priorităților, PR (Priority Resolver).

Odată acceptată de către microprocesor, cererea de întrerupere se introduce în registrul întreruperilor în servire, ISR (In Service Register). Cu ajutorul registrului de mascare a întreruperilor, IMR (Interrupt Mask Register), se poate inhiba prin program oricare din cele 8 linii de întrerupere. Solicitățile apărute pe liniile mascate nu vor mai fi introduse în ISR, deci nu vor mai fi transmise nici microprocesorului.

În momentul în care IRR conține cel puțin o solicitare de întrerupere, prin intermediul blocului logică de comandă se lansează o cerere de întrerupere către UCP (pe linia INT). În același timp se rezolvă prioritățile, iar înscrierea cererii celei mai prioritare în ISR se face numai în momentul primirii confirmării de la microprocesor ($\overline{INTA} = 0$).

Un dispozitiv 8259/8259A primește cuvinte de comandă sau transmite informații de la/spre microprocesor pe liniile $D_7 \div D_0$, prin intermediul blocului tampon pentru magistrala de date (Data Bus Buffer). Aceste linii sunt bidirecționale și au facilitatea TS, fapt ce permite conectarea PIC, ca dispozitiv I/E, la magistrala de date a sistemului. Controlul operațiilor de citire/scriere se face prin intermediul blocului logică de citire/scriere (Read/Write Logic). Linia \overline{CS} (Chip Select), în conjuncție cu activarea uneia din liniile \overline{RD} sau \overline{WR} , permite UCP să controleze transferul de informații de la/spre PIC, prin orientarea corespunzătoare a tamponului

magistralei de date. Linia A_0 , conectată de obicei la linia cu același nume a magistralei de adrese, împreună cu o parte din liniile de date, permit selectarea diferitelor registre interne ale controlerului.

Pentru realizarea de sisteme de întreruperi cu mai mult de 8 niveluri, dispozitivele 8259/8259A posedă o logică de conectare în cascadă (Cascade Buffer/Comparator), prin intermediul căreia pot fi interconectate mai multe circuite PIC. Aceasta se face direct, prin liniile $CAS_2 \div CAS_0$ (CASCade), care au rol de intrare sau ieșire, în funcție de starea liniei \overline{SP}/EN (Slave Program/ENable).

Linia \overline{SP}/EN are dublă funcționalitate numai la dispozitivele 8259A; funcția de validare asigură un transfer corect de informații între UCP și PIC în sistemele complexe, care necesită bufferizarea magistralei de date. În acest caz, \overline{SP}/EN este linie de ieșire și realizează activarea transferului (ENable) de la PIC prin buffer. La sisteme fără bufferizare este activă opțiunea \overline{SP} , atât la 8259A cât și la 8259. Aceasta permite stabilirea prin hardware a rolului fiecărui dispozitiv PIC într-o structură SINT de tip "master-slave": un PIC coordonator (sau master) are linia \overline{SP} conectată la "1" logic, iar celelalte dispozitive PIC (maximum 8), subordonate, au linia \overline{SP} conectată la masă ("0" logic). Linia INT a unui controler slave se conectează la una din intrările $IR_0 \div IR_7$ de la master, iar linia INT a controlerului master se conectează direct la linia de întreruperi a microprocesorului.

Pentru a funcționa corect, dispozitivele 8259/8259A trebuie să fie mai întâi inițializate de către microprocesor, prin înscrierea unor cuvinte de comandă.

Inițializarea PIC 8259/8259A. Cazul sistemelor cu microprocesoare 8080/8085

În fig.4.29 sunt prezentate secvențele de inițializare a celor două tipuri de controlere. Ambele dispozitive necesită cuvinte de inițializare formate din 9 biți: un bit "transmis" pe linia de adresă A_0 și opt biți transmiși pe magistrala de date. Dispozitivul 8259 necesită maximum trei cuvinte de inițializare, notate ICW1, ICW2 și ICW3 (Initialisation Command Words), iar 8259A are nevoie de cel mult patru cuvinte: primele trei au aproximativ aceeași semnificație ca și cele de la 8259, dar apare în plus ICW4. Dacă la 8259A se poziționează $D_0=IC4=0$ în ICW1, aceasta înseamnă că nu se utilizează ICW4 și atunci funcționarea devine identică cu cea a circuitului 8259.

Primele două cuvinte de inițializare, ICW1 și ICW2, stabilesc adresa de început a unui tabel de întreruperi pentru fiecare controler din SINT, organizat în memorie de către programator. În acest tabel se alocă un spațiu (interval) de 4 sau 8 octeți, pentru fiecare din cele 8 linii de întrerupere ($IR_7 \div IR_0$), după cum bitul $D_2=ADI$ (call ADdress Interval) este poziționat pe "1" sau pe "0" în ICW1. Rezultă astfel o dimensiune a tabelului de întreruperi de 64, respectiv 32 de octeți (consecutivi). Adresa rutinei de tratare a întreruperii pentru o linie oarecare IR_i ($i=0 \div 7$) este chiar adresa de început a intervalului i din tabel. Aceste adrese sunt multiplu de 4 sau de 8, ceea ce impune ca adresa de început a tabelului să fie un multiplu de 32, respectiv de 64.

La acceptarea unei întreruperi pe linia i , controlerul calculează cei 5 sau 6 biți mai puțin semnificativi ai adresei de tratare ca fiind $i \times 4$, respectiv $i \times 8$. Ei sunt inserați automat de către controler, alături de ceilalți 11, respectiv 10 biți stabiliți la inițializarea circuitului. Biții $A_7 \div A_5$, respectiv $A_7 \div A_6$ din ICW1 completează octetul mai puțin semnificativ al adresei, care se extinde apoi cu biții $A_{15} \div A_8$, definiți de ICW2.

Astfel, prin cele două cuvinte, ICW1 și ICW2, înscrise o singură dată - la inițializarea controlerului, se definesc adresele de tratare (vectorii) pentru toate cele 8 linii de întrerupere. La aceste adrese, proiectantul de sistem trebuie să prevadă instrucțiuni de salt către rutinele de tratare propriu-zise. Bitul $D_1=S$ (Single) din ICW1 indică, pe "1", faptul că există un singur PIC în SINT; dacă $S=0$, atunci sunt mai multe controlere, conectate în cascadă, într-un sistem master-slave.

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

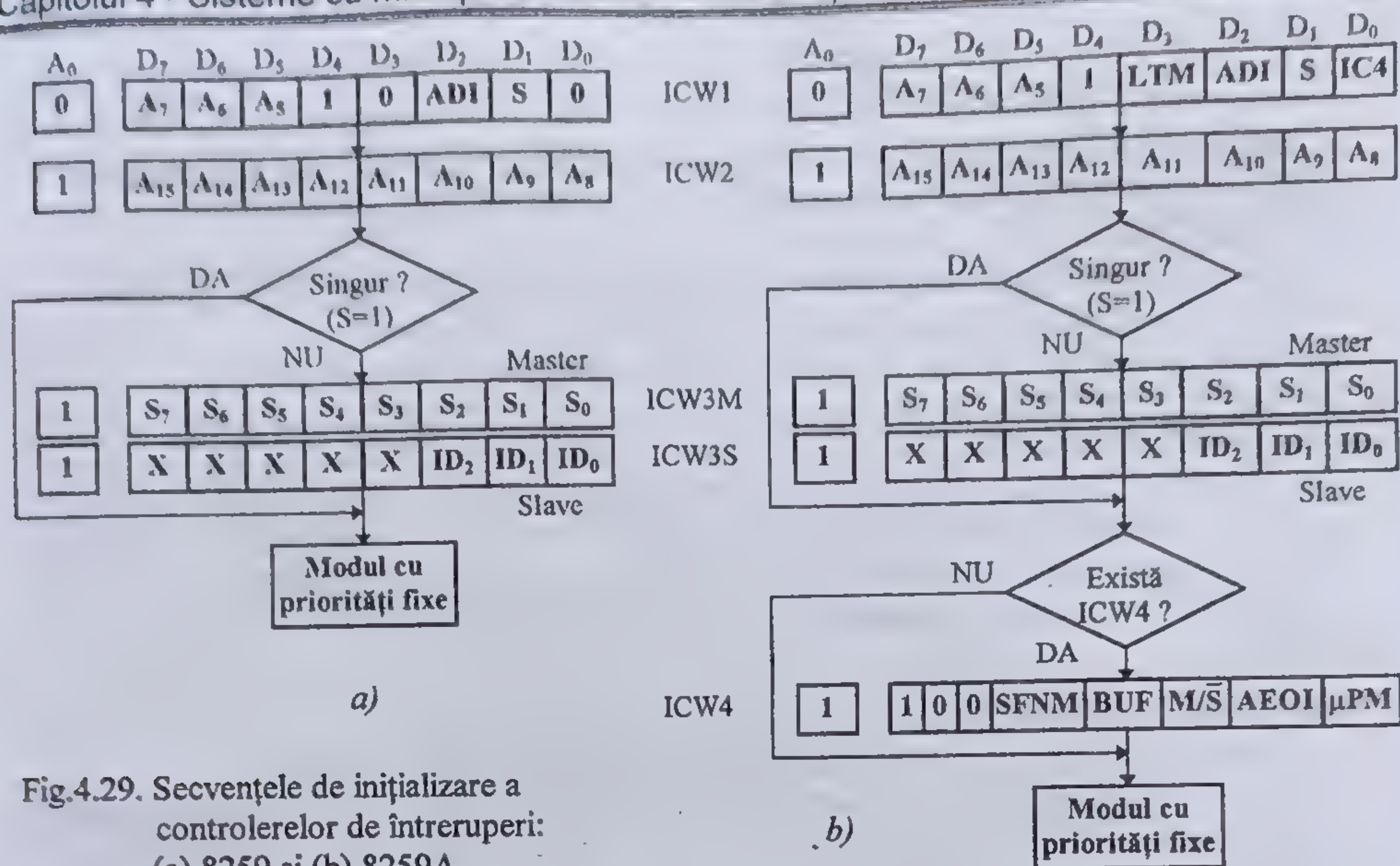


Fig.4.29. Secvențele de inițializare a controlerelor de întreruperi: (a) 8259 și (b) 8259A

La dispozitivul 8259A poate fi programat, prin bitul $D_3=LTM$, modul de prezentare a cererilor de întrerupere pe liniile $IR_0 \div IR_7$: dacă $LTM=0$ - cele 8 linii sunt active pe front crescător (Edge Triggered Mode), iar dacă $LTM=1$ - sunt active pe nivel logic "1" (Level Triggered Mode). În ambele cazuri trebuie menținut nivelul logic "1" până când microprocesorul obține adresa rutinei de tratare. La PIC 8259, liniile $IR_0 \div IR_7$ sunt active numai pe nivel ridicat.

Conform organigramelor din fig.4.29, cuvintele ICW1 și ICW2 sunt suficiente pentru ca un SINT format dintr-un singur dispozitiv 8259 (sau 8259A, dacă $IC4=0$) să poată opera în *modul cu priorități fixe* (Fully Nested Mode), imediat după inițializare.

În cazul în care există mai multe PIC în SINT (sisteme cu mai mult de 8 niveluri de întrerupere), este necesar și un al treilea cuvânt de inițializare, ICW3. Pentru controlerul master acesta este notat cu ICW3M (v.fig.4.29), în care prin $S_i=1$ ($i=0 \div 7$) se specifică faptul că pe linia IR_i este conectat un controler slave. Pentru un controler slave, cel de-al treilea cuvânt este notat cu ICW3S; biții $ID_2 \div ID_0$ poartă numele de *cod de recunoaștere* (IDentify) și reprezintă, codificat pe 3 biți, numărul liniei IR_i de la master pe care este conectat controlerul slave.

Numai la dispozitivul 8259A, dacă $IC4=1$ în ICW1, se utilizează și un al 4-lea cuvânt de inițializare, ICW4. Prin intermediul acestuia se pot utiliza opțiunile suplimentare față de 8259, cu care este prevăzut acest tip de controler, după cum urmează:

- bitul $D_0=\mu PM$ (Microprocessor Mode) - specifică tipul de microprocesor al UCP: dacă $D_0=0$, 8259A este setat să funcționeze în sisteme cu 8080/8085, iar când $D_0=1$ - în sisteme cu 8086.
- bitul $D_1=AEOI$ (Automatic End Of Interrupt) - când are valoarea "1", permite resetarea automată a bitului din ISR corespunzător cererii în curs de servire, în momentul când microprocesorul recepționează adresa completă a rutinei de tratare a întreruperii (achitare automată a întreruperii). Dacă $AEOI=0$, pentru resetarea bitului din ISR este necesar un cuvânt special de comandă de la microprocesor, EOI (End Of Interrupt), care confirmă astfel controlerului încheierea servirii întreruperii (achitare explicită a întreruperii);

- biții D_3 (BUF) și D_2 (M/\bar{S}) sunt legați funcțional de cazul sistemelor la care magistrala de date este bufferizată și când linia \overline{SP}/EN este folosită ca linie de comandă a bufferului, în dialogul controlerului cu UCP. Deoarece, în acest caz, linia \overline{SP}/EN pierde opțiunea \overline{SP} , poziția de master sau slave a unui PIC în SINT se programează prin D_2 : "1" pentru master și "0" pentru slave.

- bitul D_4 =SFNM (Special Fully Nested Mode) permite acceptarea unei cereri de întrerupere de la un PIC slave, chiar și atunci când există deja o cerere în curs de servire, primită anterior, de la același controler. În mod normal (SFNM=0), o nouă solicitare de la un slave din SINT este blocată de master până la achitarea celei în curs de servire. Acest regim special se obține transmițând un cuvânt ICW4 cu $D_4=1$ numai la PIC-master. În acest caz, SINT se comportă ca și cum ar fi format dintr-un singur controler, cu 15 linii de întrerupere. La revenirea din rutina corespunzătoare unei linii de la PIC-slave, se achită mai întâi întreruperea de la PIC-slave și, numai dacă nu mai este o alta în curs de tratare la slave, se achită și la PIC-master.

Dacă $IC4=0$ în ICW1, atunci funcțiile asociate biților din ICW4 sunt cele corespunzătoare lui $ICW4=00h$.

Modurile de operare ale dispozitivelor 8259/8259A

După inițializare, se poate alege unul din următoarele moduri de operare:

- modul cu priorități fixe** (Fully Nested mode) - în acest mod se intră imediat după inițializare, când se șterg registrele IMR, IRR și ISR, liniei IR_0 i se atribuie prioritate maximă, iar liniei IR_7 prioritate minimă.
- modul cu rotirea priorităților** (Rotating mode), prin care se poate asigura o tratare echiprioritară a liniilor de întrerupere. La acest mod de operare există două variante: 1) *cu rotire normală* (Auto mode), când rotirea priorităților se face ciclic, în ordine naturală (de exemplu, după tratarea întreruperii pe o linie oarecare, aceasta capătă prioritate minimă, celelalte rotindu-se în mod corespunzător), respectiv 2) *cu rotire specifică* (Specific mode), când se poate desemna prin program linia care va primi prioritate minimă.
- modul de mascare specială** (Special Mask mode); într-o rutină de întrerupere poate fi necesară inhibarea cererilor de pe nivelurile inferioare în anumite porțiuni (modul normal), dar și validarea unora dintre ele, în alte porțiuni. Modul de mascare specială permite, pentru liniile mascate, inhibarea cererilor suplimentare de pe aceeași linie, dar validează solicitările de pe oricare din celelalte niveluri nemascate, indiferent dacă sunt superioare sau inferioare.
- modul cu interogare** (Polled mode), în care se trece la tratarea prin interogare a dispozitivelor I/E, renunțându-se la tratarea vectorizată, prin întreruperi. În acest mod, microprocesorul nu mai primește cereri de întrerupere, ci trebuie să lanseze periodic proceduri de interogare a controlerelor de întreruperi din sistem. Acestea se transformă în simple sisteme de înregistrare a cererilor de întrerupere, furnizând - numai la cererea microprocesorului - informații despre existența solicitărilor de "întrerupere", într-un cuvânt de forma următoare:

Dacă există o solicitare de întrerupere, $D_7=I=1$, atunci activarea semnalului \overline{RD} ($\overline{RD}=0$, $A_0=x$) este interpretată de 8259/8259A ca și un răspuns la cerere (\overline{INTA}) și determină setarea în ISR a bitului corespunzător liniei de prioritate maximă, având codul precizat de biții W_2+W_0 .

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
I	-	-	-	-	W_2	W_1	W_0

După inițializare, dispozitivele 8259/8259A intră automat în modul cu priorități fixe (v. fig.4.29). Funcționarea în celelalte moduri menționate mai sus se face prin transmiterea de către microprocesor a unor *cuvinte de comandă de operare* (Operation Command Words) corespunzătoare.

Ambele dispozitive folosesc aceleași tipuri de cuvinte de operare, notate OCW1, OCW2 și OCW3. Aceste cuvinte au tot o structură de 9 biți, ca și cuvintele de inițializare.

OCW1 - asigură mascarea unei sau mai multor linii de întrerupere și are forma următoare:

A_0 D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 $M_i=1$ ($i=7+0$) asigură mascarea liniilor de întrerupere IR_i , atât în modul normal, cât și în cel special.

1	M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0
---	-------	-------	-------	-------	-------	-------	-------	-------

OCW2 conține următoarele informații:

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	R	SEOI	EOI	0	0	L_2	L_1	L_0

codul liniei, în "Specific mode"

0	0	1	Non specific End Of Interrupt	Achitare întrerupere
0	1	1	Specific End Of Interrupt	
1	0	1	Rotire odată cu resetarea bitului "1" de prioritate maximă din ISR	Rotire automată
1	0	0	Setare mod rotire în "Automatic EOF"	
0	0	0	Resetare mod rotire în "Automatic EOF"	
1	1	1	Rotire odată cu resetarea bitului "1" din ISR specificat de $L_2 L_1 L_0$	Rotire specifică
1	1	0	Stabilire nivel de prioritate minimă (cel specificat de $L_2 L_1 L_0$)	
0	1	0	Nici o operație	

Într-un SINT cu circuite 8259A programate cu bitul $AEIOI=1$ în ICW4, funcționând în modul cu priorități fixe, OCW2 nu este necesar; bitul din ISR este resetat automat, imediat după preluarea vectorului de întrerupere. În schimb, există comenzi OCW2 de trecere, respectiv de revenire, în/din modul cu rotirea automată a priorităților.

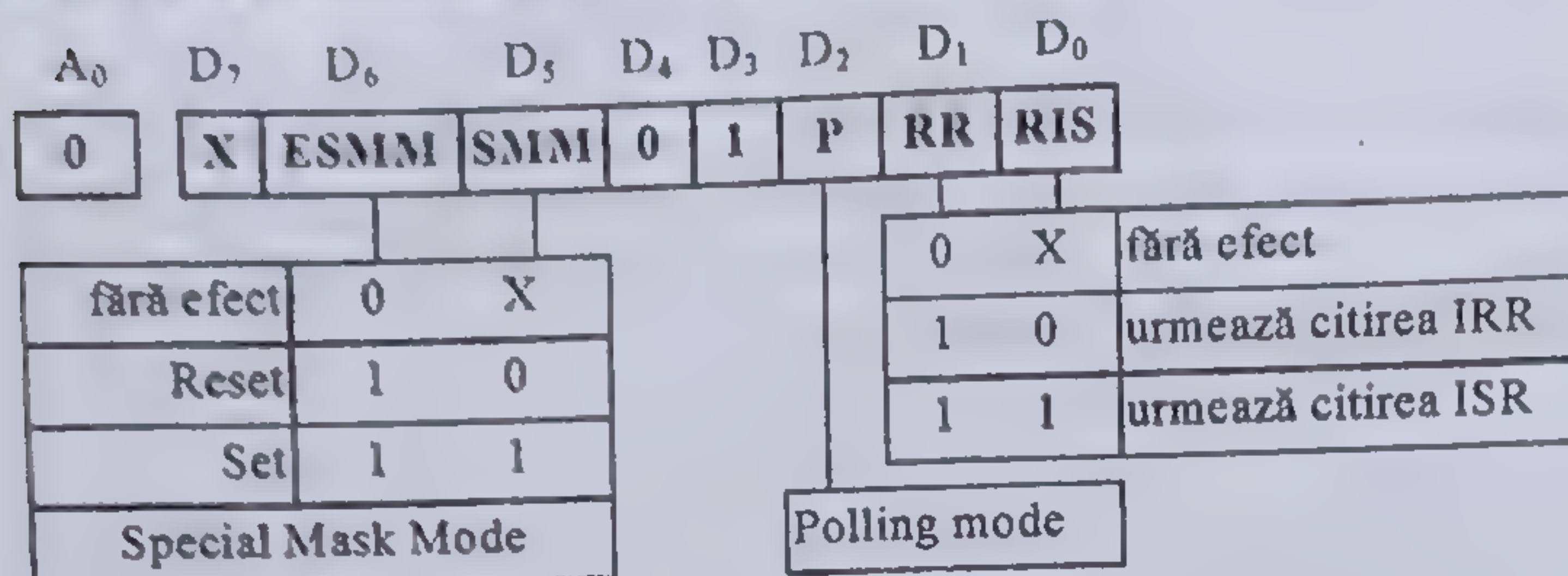
Dacă bitul $AEIOI=0$ în ICW4 sau dacă nu se utilizează ICW4 (este și cazul SINT cu 8259), atunci bitul din ISR corespunzător cererii de întrerupere în curs de servire trebuie resetat explicit, înainte de revenirea din întrerupere; aceasta se poate face cu un cuvânt OCW2 având bitul $D_5=EOI=1$ (non-specific End Of Interrupt). În modul cu priorități fixe sau cu rotirea normală a priorităților, bitul care trebuie resetat este identificat automat ca fiind bitul "1" de prioritate maximă din ISR. În modul cu rotirea specifică a priorităților, rutina de tratare trebuie să precizeze explicit care este bitul din ISR ce trebuie resetat, prin liniile L_2+L_0 , cu ajutorul unui cuvânt OCW2 cu $D_6=SEOI=1$ și $EOI=1$ (Specific End Of Interrupt).

Comenzile de achitare a întreruperilor (EOI și SEOI) se pot da independent sau împreună cu comenzile de rotire a priorităților. În ultimul caz, în cuvântul OCW2 transmis din rutina de tratare a întreruperii, se poziționează bitul $D_7=R=1$ (Rotating mode). Acest bit, împreună cu $SEOI=0$ și $EOI=1$, conduce la varianta de *rotire normală*, când se resetează bitul "1" având prioritate maximă în ISR. Dacă $R=1$, $SEOI=1$ și $EOI=1$ se obține cea de-a doua variantă, de rotire și achitare specifică. În această variantă se resetează bitul "1" din ISR corespunzător liniei având codul precizat de L_2+L_0 , care totodată capătă prioritate minimă.

Prioritățile pot fi rotite și fără achitarea întreruperii, dar numai specific: $R=1$ și $SEOI=1$, $EOI=0$, fac ca linia având codul precizat de L_2+L_0 să devină cea mai puțin prioritară.

Biții $D_4=D_3=0$, împreună cu $A_0=0$, asigură recunoașterea cuvântului OCW2 de către controler.

OCW3 are următoarea structură:



Recunoașterea cuvântului de către controler se face prin A₀=0 și D₄=0, D₃=1.

Biți D₅ (SMM) și D₆ (Erase SMM) permit setarea sau ștergerea modului special de mascare. Bitul D₂ (P) asigură trecerea la modul cu interogare (D₂=P=1) a dispozitivelor I/E.

În plus, OCW3 permite citirea de către microprocesor a conținutului registrelor IRR sau ISR. În acest scop, microprocesorul trebuie să înscrie un cuvânt OCW3 cu D₁=RR=1 și cu D₀=RIS egal cu "0" sau cu "1", după cum se dorește citirea IRR, respectiv a ISR.

Citirea propriu-zisă se face la primul impuls \overline{RD} de după transmiterea cuvântului de comandă, impuls generat de o instrucțiune IN PIC, unde PIC este adresa de bază a controlerului de întreruperi (cu A₀=0). O astfel de procedură permite utilizatorului să testeze corecta funcționare a controlerului și să ia decizii în consecință. Registrul IMR poate fi citit în orice moment, printr-o instrucțiune IN PIC+1 (cu A₀ = 1).

4.4.2.2. Organizarea SINT cu PIC 8259/8259A

Facilitățile pe care le au controlerele programabile 8259 și 8259A permit organizarea unor sisteme puternice de gestionare și servire a cererilor multiple de întrerupere. În acest mod cresc posibilitățile de lucru în timp real, chiar la sisteme cu procesoare relativ modeste cum sunt 8080 și 8085. Principiile de organizare a SINT cu aceste microprocesoare rămân în esență valabile și pentru alte tipuri de procesoare.

La proiectarea unui SINT trebuie să se trateze atât aspectele hardware cât și software, în corespondență cu caracteristicile controlerelor și cu cerințele aplicației.

Fiind cunoscut numărul surselor de întreruperi, se definește structura HW a SINT având în vedere următoarele:

- necesarul de controlere și interconectarea acestora;
- conectarea corectă la magistralele unității centrale;
- adresarea și selecția fiecărui PIC din SINT, în corespondență cu spațiul de adresare disponibil.

Partea SW implică programarea corespunzătoare a PIC din SINT.

Pentru explicitarea aspectelor enunțate mai sus se va apela la următorul exemplu.

Exemplu: Să se organizeze un SINT cu 15 niveluri, fiind cunoscut faptul că tabelul cu vectorii de întreruperi va fi amplasat în memorie cu începere de la adresa 1000h, va avea intervale de 4 octeți, iar spațiul I/E ocupat este definit, prin decodificare, de adresele A₁₅+A₁₂.

Din specificațiile problemei rezultă că sunt necesare două controlere, interconectate într-o structură "master-slave". Dispozitivul slave se conectează la master pe una din intrările IR_i (i=7+0). Deoarece PIC-master va lucra cu priorități fixe și va gestiona cererile de prioritate ridicată, controlerele subordonate din SINT se conectează de obicei la liniile de prioritate mică ale masterului. În cazul de față se va considera conectarea la linia IR₇.

În fig.4.29 este prezentată structura HW a SINT într-un sistem cu microprocesor 8085; pentru simplificare s-au considerat numai conexiunile relevante din punctul de vedere al tratării

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

Întreperilor. Trecerea la UCP cu 8080 nu prezintă dificultăți, dacă se au în vedere cele prezentate în §4.1.

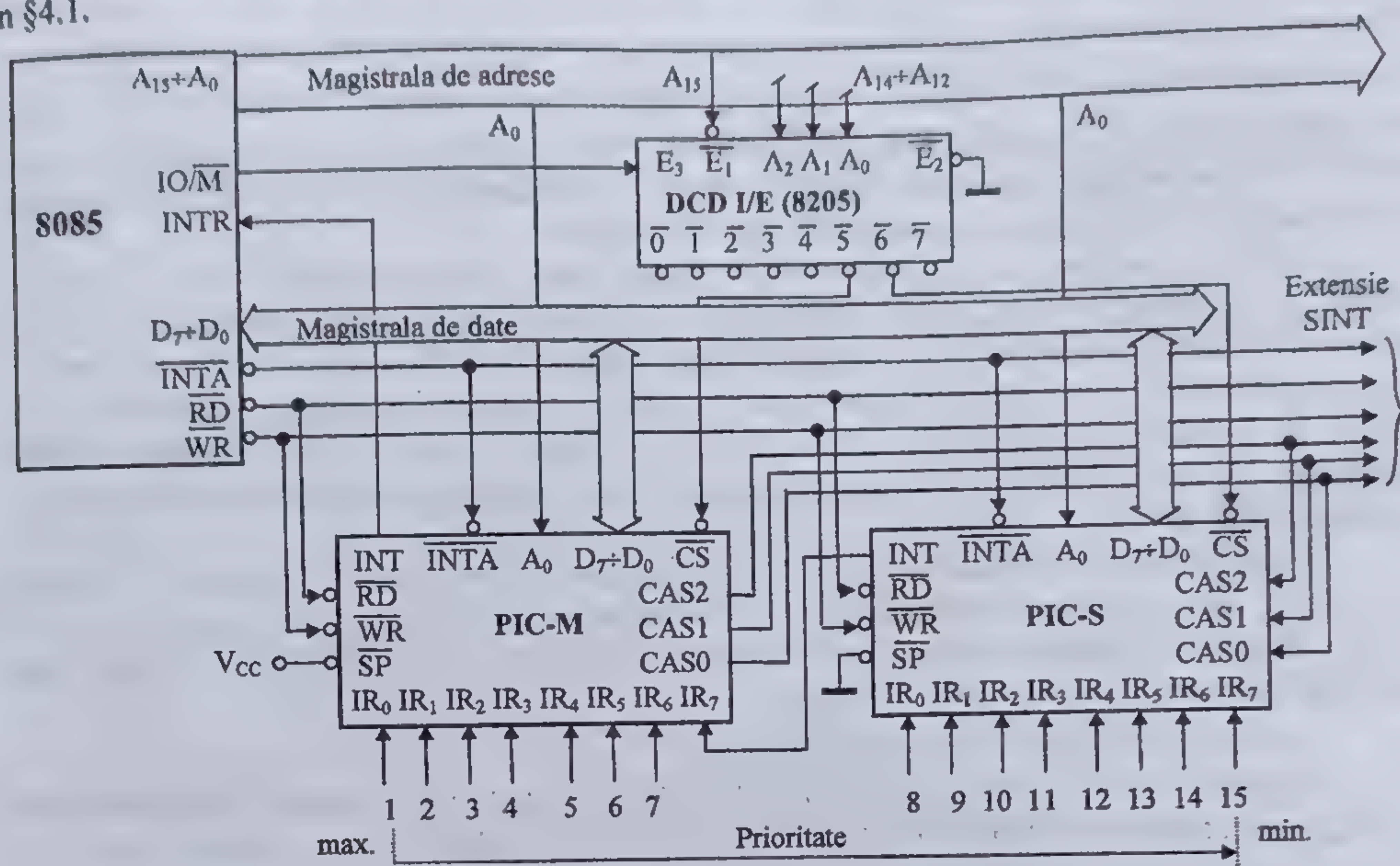


Fig.4.29. Structura hardware a SINT cu 15 niveluri

Din modul de conectare a DCD I/E (8205) rezultă că PIC-M are adresa 50h, iar PIC-S adresa 60h. Definirea rolului fiecărui controler în SINT, master sau slave, se face prin fixarea la V_{CC} , respectiv la masă, a liniei \overline{SP} . Organizarea SINT impune conectarea celor două controlere prin intermediul liniilor $CAS_2 \div CAS_0$, care formează o magistrală de cascader. Se observă că expandarea SINT este ușor de realizat prin adăugarea de noi PIC-slave (încă maximum 7). Toate controlerele din SINT se conectează în paralel la liniile de control \overline{INTA} , $\overline{I/OR}$ și $\overline{I/OW}$.

Lansarea unei cereri de întrerupere către UCP se face numai de către PIC-M, pe linia INT. Dialogul între microprocesor și controlere în regimul de întreruperi are loc în mod diferit, în funcție de poziția acestora în SINT, după cum se va arăta în continuare. Se consideră ambele controlere inițializate (v. §4.4.2.1).

Funcționarea SINT

Comportarea SINT, începând de la apariția unei cereri de întrerupere și până la terminarea tratării rutinei corespunzătoare, diferă în funcție de circuitul la care este prezentată solicitarea: la PIC-M sau la PIC-S. Controlerul master este capabil să distingă dacă o solicitare provine sau nu de la un controler slave (vezi ICW3M în §4.4.2.1) și să acționeze în mod diferențiat, așa cum se va arăta în cele ce urmează.

Tratarea solicitărilor provenite de la PIC-M

După rezolvarea priorităților, PIC-M lansează o solicitare de întrerupere către UCP, pe linia INT. În momentul în care microprocesorul acceptă această cerere, intră în regimul de întrerupere. În primul ciclu mașină de întrerupere (ciclul INA în tab.4.4 - v. §4.2.1), 8085 generează un prim impuls \overline{INTA} . La primirea acestui semnal de confirmare, PIC-M forțează pe magistrala de date opcodul instrucțiunii CALL $addr_1$ (CDh), care este citit de către microprocesor.

În același timp, PIC-M setează în ISR bitul corespunzător cererii acceptate pentru servire și depune, pe liniile CAS_2+CAS_0 , codul i corespunzător liniei pe care a primit solicitarea de întrerupere, IR_i . În urma decodificării opcodului citit, microprocesorul lansează încă două cicluri succesive de citire (v. ciclul INA, §4.2.1), în care 8085 mai generează câte un impuls \overline{INTA} . După recepționarea celui de-al doilea impuls \overline{INTA} , PIC-M furnizează pe magistrala de date octetul inferior, apoi pe cel superior al adresei din tabelul de întrerupere, corespunzător liniei pe care a apărut solicitarea (vezi discuția de la ICW1 și ICW2, din §4.4.2.1).

După citirea completă a instrucțiunii CALL $addr_i$, microprocesorul trece la execuția acesteia: se salvează în stivă adresa instrucțiunii următoare din programul întrerupt și apoi se realizează un salt la adresa $addr_i$. La această adresă se poate găsi chiar rutina de tratare, dacă este suficient de scurtă să încapă în cei maximum 8 octeți rezervați la inițializare (vezi bitul ADI în ICW1). De obicei, la adresa respectivă se află o instrucțiune de salt la adresa rutinei de tratare propriu-zise. În acest caz se utilizează intervale de 4 octeți, din care doar câte 3 sunt ocupați de instrucțiunile de salt.

Microprocesorul execută rutina de tratare, dar, înainte de revenirea în programul întrerupt, trebuie să anunțe controlerul master despre terminarea servirii (excepție face cazul 8259A cu AEOI=1 în ICW4). Aceasta se face printr-un cuvânt OCW2 cu EOI sau SEOI pe "1" logic, astfel încât bitul corespunzător din ISR va fi resetat.

Tratarea solicitărilor provenite de la PIC-S

În cazul acceptării unei cereri de la PIC-S, funcționarea SINT este similară în prima parte cu cea prezentată anterior, având loc următoarea secvență de operații:

$$a) \text{ PIC-S} \xleftarrow{IR_i} \text{ disp I/E}$$

$$b) \text{ PIC-M} \xleftarrow{IR_7} \text{ PIC-S}$$

$$c) \mu P \xleftarrow{INT} \text{ PIC-M}$$

$$d) \mu P \xrightarrow{\overline{INTA}} \begin{cases} \text{PIC-M} \\ \text{PIC-S} \end{cases}$$

Odată cu apariția primului impuls \overline{INTA} , la PIC-M se setează bitul D_7 în ISR, iar la PIC-S se setează bitul D_1 .

$$e) \mu P \xleftarrow{CDh(CALL)} \text{ PIC-M} \xrightarrow{111_2(7)} CAS_2+CAS_0;$$

Ca și mai înainte, PIC-M răspunde microprocesorului cu opcodul instrucțiunii CALL $addr_i$ și depune, pe liniile CAS_2+CAS_0 , codul liniei pe care a ajuns solicitarea la master (IR_7 în acest caz). Microprocesorul decodifică opcodul instrucțiunii CALL, iar PIC-S compară codul primit de la PIC-M pe liniile CAS_2+CAS_0 cu codul propriu, programat la inițializare (vezi biții ID_2+ID_0 din ICW3S în §4.4.2.1).

$$f) \mu P: \text{decodificare } CDh; \text{ PIC-S} \xrightarrow{ID_2+ID_0} CAS_2+CAS_0$$

Numai PIC-S al cărui cod propriu ID_2+ID_0 coincide cu datele de pe CAS_2+CAS_0 continuă dialogul cu microprocesorul, la următoarele două impulsuri \overline{INTA} ; PIC-M nu mai intervine pe magistrala de date.

g) $\mu P \xrightarrow{\overline{INTA}} PIC-S; \mu P \xleftarrow{LSB} PIC-S$, unde $LSB = LOW(addr_i)$

h) $\mu P \xrightarrow{\overline{INTA}} PIC-S; \mu P \xleftarrow{MSB} PIC-S$, unde $MSB = HIGH(addr_i)$

După citirea completă a instrucțiunii, se salvează pe stivă adresa de revenire în programul întrerupt (incrementarea registrului PC a fost blocată pe durata dialogului cu PIC-M și PIC-S):

i) $\mu P \xrightarrow{(PC)} STACK$

În final se face saltul la rutina de tratare a cererii de întrerupere lansată de PIC-S:

j) $\mu P \xrightarrow{addr_i} PC$

Din rutina de tratare, înainte de revenirea în programul întrerupt, trebuie transmise comenzi OCW2 (de tip EOI sau SEOI), atât la PIC-M, cât și la PIC-S.

k) $\mu P \xrightarrow{OCW2} \begin{cases} PIC-M \\ PIC-S \end{cases}$

Programarea SINT

Aceasta implică rezolvarea mai multor aspecte. În primul rând, trebuie realizată inițializarea fiecărui controler din SINT prin transmiterea, în ordine, a cuvintelor de comandă corespunzătoare: ICW1, ICW2 și ICW3 (eventual și ICW4). Nivelurile SINT care nu se utilizează, se maschează folosind un cuvânt OCW1. De asemenea, se stabilesc și celelalte cuvinte de operare necesare (OCW2, OCW3), în funcție de modul de operare utilizat.

Un alt aspect important este organizarea tabelului de întreruperi. În cazul exemplului considerat, pentru PIC-M tabelul $tabm$ va începe la adresa 1000h și va ocupa 32 (20h) octeți, deoarece se utilizează intervale de 4 octeți pentru fiecare din cele 8 linii. Se observă că această adresă respectă condiția de a fi multiplu de 32, întrucât are ultimii 5 biți egali cu 0. Pentru PIC-S, tabelul $tabs$ va avea aceeași dimensiune și va fi amplasat în continuarea celui de la master (1000h+20h=1020h).

Scrierea rutinelor de tratare a cererilor de întrerupere completează programarea SINT. În afară de acțiunile de tratare propriu-zisă a solicitărilor, specifice aplicației, o rutină de întrerupere trebuie să conțină salvarea, respectiv refacerea contextului programului întrerupt, comenzile de achitare a întreruperii (EOI sau SEOI), precum și revenirea din întrerupere (RET), precedată de revalidarea întreruperilor microprocesorului pe linia INT, cu o instrucțiune EI (Enable Interrupt).

În corpul unei rutine de întrerupere este posibil, în funcție de necesități, ca după efectuarea unor activități critice din punctul de vedere al timpului de răspuns (*secțiunea neîntreruptibilă*), să se activeze mecanismul de întrerupere (EI), pentru luarea în considerare a solicitărilor de prioritate superioară celei în curs de tratare (*secțiunea întreruptibilă*). Comenzile de achitare a întreruperii (OCW2 de tip EOI) afectează lanțul de priorități al controlerelor de întrerupere. Pentru a lăsa timp suficient controlerului să interpreteze corect aceste comenzi și lanțului de priorități să se stabilizeze, este necesar ca astfel de comenzi să se dea numai din secțiuni neîntreruptibile. Din acest motiv, înainte de înscrierea comenzii EOI, este dezactivat sistemul de întrerupere al microprocesorului cu o instrucțiune DI.

Rezultă următoarele secvențe de program:

; INIȚIALIZARE PIC-M

```

ICW1M EQU 00010100b ; 4 octeți/interval, master-slave, adresa 1000h
                        ; adresa 1000h
ICW2M EQU 10h ;
ICW3M EQU 10000000b ; Există slave conectat pe IR7 la master.
PIC_M EQU 50h ; Adresa controlerului master (de la DCD 8205)

MVI A,ICW1M ; Se încarcă în A cuvântul de comandă ICW1 pentru master.
OUT PIC_M ; Se înscrie pe adresa master-ului (cu A0=0).
MVI A,ICW2M ; Se încarcă în A cuvântul de comandă ICW2 pentru master.
OUT PIC_M+1 ; Se înscrie pe adresa master-ului (cu A0=1).
MVI A,ICW3M ; Se încarcă în A cuvântul de comandă ICW3 pentru master.
OUT PIC_M+1 ; Se înscrie pe adresa master-ului (cu A0=1).

```

; INIȚIALIZARE PIC-S

```

ICW1S EQU 00110100b ; 4 octeți/interval, master-slave, adresa 1020h
                        ; adresa 1020h
ICW2S EQU 10h ;
ICW3S EQU 00000111b ; Este conectat la master pe linia IR7.
PIC_S EQU 60h ; Adresa controlerului slave (de la DCD 8205)

MVI A,ICW1S ; Se încarcă în A cuvântul de comandă ICW1 pentru slave.
OUT PIC_S ; Se înscrie pe adresa slave-ului (cu A0=0).
MVI A,ICW2S ; Se încarcă în A cuvântul de comandă ICW2 pentru slave.
OUT PIC_S+1 ; Se înscrie pe adresa slave-ului (cu A0=1).
MVI A,ICW3S ; Se încarcă în A cuvântul de comandă ICW3 pentru slave.
OUT PIC_S+1 ; Se înscrie pe adresa slave-ului (cu A0=1).

```

; PROGRAMARE MĂȘTI - Implicit, după inițializare, toate nivelurile sunt validate.

; Dacă se dorește ca, pentru început, să nu fie acceptate întreruperile de pe anumite linii, acestea se maschează folosind cuvinte de comandă de tip OCW1:

```

OCW1M EQU 01000000b ; Se maschează linia IR6 de la master
OCW1S EQU 11110000b ; și liniile IR7÷IR4 de la slave.

MVI A,OCW1M ; Se încarcă în A cuvântul de mască OCW1 pentru master.
OUT PIC_M+1 ; Se înscrie pe adresa master-ului (cu A0=1).
MVI A,OCW1S ; Se încarcă în A cuvântul de mască OCW1 pentru slave.
OUT PIC_S+1 ; Se înscrie pe adresa slave-ului (cu A0=1).

```

; ORGANIZAREA TABELULUI DE INTRERUPERI

```

ORG 1000h ; tabm începe la 1000h.
tabm: JMP trat0 ; Aici se sare la apariția unei solicitări de întrerupere pe linia IR0 de la
NOP ; master. JMP trat0 are 3 octeți, NOP are 1 octet, în total sunt 4 octeți.
JMP trat1 ; Salt la rutina de tratare a solicitării pe linia IR1 de la master.
NOP
JMP trat2 ; Se ajunge aici ca urmare a faptului că în ciclurile de întrerupere,
NOP ; PIC-M furnizează microprocesorului instrucțiunea CALL trat2.
; .....
JMP trat6 ; Salt la rutina de tratare a solicitării pe linia IR6 de la master.
NOP
JMP eroare ; EROARE: la o solicitare pe linia IR7 de la master nu se ajunge aici,
NOP ; ci în tabelul de la slave (adresa de la CALL e furnizată de PIC-S).
tabs: JMP trat7 ; Salt la rutina de tratare a solicitării pe linia IR0 de la slave.
NOP
JMP trat8 ; Salt la rutina de tratare a solicitării pe linia IR1 de la slave.
NOP
; .....
JMP trat14 ; Salt la rutina de tratare a solicitării pe linia IR7 de la slave

```


Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

; STRUCTURA UNEI RUTINE DE TRATARE A ÎNTRERUPERII DE LA PIC-M (trat0+trat6)

EOI EQU 00100000b ; Definește un cuvânt de comandă OCW2 de tip EOI.
trat0: ; Sau trat1 ... trat6

PUSH PSW ; Salvează pe stivă acumulatorul, flag-urile,
PUSH B ; precum și conținutul celorlalte registre modificate de subrutină.
PUSH D
PUSH H

..... ; Tratare specifică dispozitivului solicitant (secțiune neîntreruptibilă).

EI

..... ; Tratare specifică dispozitivului solicitant (secțiune întreruptibilă).

MVI A,EOI ; Încarcă în A cuvântul de achitare a întreruperii (conservă prioritățile)
DI ; Invalidare întreruperi - intrare în secțiunea critică.
OUT PIC_M ; Achită întreruperea - numai la PIC-master.
POP H ; Reface de pe stivă contextul programului întrerupt, salvat la intrare.
POP D
POP B
POP PSW
EI ; Revalidează întreruperile.
RET ; Revine în programul întrerupt.

; STRUCTURA UNEI RUTINE DE TRATARE A ÎNTRERUPERII DE LA PIC-SLAVE (trat7+trat14)

trat7: ; similar - trat8, ..., trat14

PUSH PSW ; Salvează pe stivă acumulatorul, flag-urile,
PUSH B ; precum și conținutul celorlalte registre modificate de subrutină.
PUSH D
PUSH H

MVI A,EOI ; Încarcă în A cuvântul de achitare a întreruperii (conservă prioritățile).
OUT PIC_M ; Achită întreruperea - mai întâi la PIC-master.

..... ; Tratare specifică dispozitivului solicitant (secțiune neîntreruptibilă).

EI

..... ; Tratare specifică dispozitivului solicitant (secțiune întreruptibilă).

MVI A,EOI ; Încarcă în A cuvântul de achitare a întreruperii (conservă prioritățile).
DI ; Invalidare întreruperi - intrare în secțiunea critică.
OUT PIC_S ; Achită întreruperea - și la PIC-slave.
POP H ; Reface de pe stivă contextul programului întrerupt, salvat la intrare.
POP D
POP B
POP PSW
EI ; Revalidează întreruperile.
RET ; Revine în programul întrerupt.

Dacă se utilizează circuite 8259, în rutinele de tratare a cererilor de la PIC-S, pentru a permite PIC-M să preia solicitări de la nivelurile superioare ale PIC-S, este preferabil să se transmită controlerului master comanda EOI chiar la începutul rutinei de întrerupere (ca în exemplul de mai sus). Se obține astfel o funcționare similară cu cea din modul SFNM (vezi IC4), de la circuitele 8259A. Prin comenzi specifice de operare se poate trece de la modul cu priorități fixe la oricare din celelalte moduri, în funcție de necesități.

Spre exemplu:

: STABILIREA MODULUI SPECIAL DE MASCARE

: În rutina de pe nivelul 10 se vor accepta și solicitări de pe nivelurile 11+14.

```

MVI    A, 00001000b    ; Mascare nivel 10 (IR3 de la PIC-slave)
OUT    PIC_S+1          ; Înscrie comanda de mascare (OCW1).
MVI    A, 01101000b    ; Comandă intrarea în modul special de mascare (OCW3).
OUT    PIC_S            ;

```

: CITIREA REGISTRULUI ISR DIN PIC-M

```

MVI    A, 00001011b    ; Comandă citire ISR.
OUT    PIC_M            ; Înscrie OCW3 în PIC_M.
IN     PIC_M            ; (A) ← (ISR)

```

4.5. Introducerea timpului în prelucrarea informației

Așa cum s-a arătat și în §2.4, pentru introducerea timpului în prelucrarea informației în sistemele cu microprocesoare sunt disponibile circuite specializate, programabile. Acestea determină creșterea complexității și a prețului de cost al sistemelor, dar degreveză microprocesorul unității centrale de implementarea taskurilor de timp, indispensabile pentru o funcționare în timp real.

Unul din cele mai utilizate dispozitive dedicate acestui scop este circuitul **I8253**, realizat de firma Intel pentru a funcționa în sistemele bazate pe microprocesoarele sale de 8 biți. Datorită facilităților pe care le posedă, acesta poate fi folosit și cu alte tipuri de microprocesoare la realizarea sistemelor de introducere a timpului (SIT).

4.5.1. Circuitul de timp 8253

Acesta este un dispozitiv programabil pentru intervale de timp (PIT - *Programmable Interval Timer*) și are schema bloc din fig.4.30.

Realizat în tehnologie NMOS, sub forma unui dispozitiv cu 24 de pini, PIT 8253 este alimentat la +5V. El conține trei ceasuri programabile independente (#0, #1 și #2), realizate în jurul unor numărătoare cu decrementare (*down counters*) de 16 biți, care pot funcționa atât în binar cât și în BCD. Fiecare ceas posedă trei linii externe, de nivel TTL:

CLK - intrare de numărare, pe care se aplică impulsuri, reprezentând evenimentele externe care urmează a fi contorizate;

GATE - intrare de validare/inhibare a decrementării numărătorului sau de reîncărcare a sa cu valoarea inițială;

OUT - pin de ieșire, pentru evidențierea în exterior a funcționării canalului de timp.

Frecvența maximă acceptată pe liniile CLK i ($i = 0, 1, 2$) este de

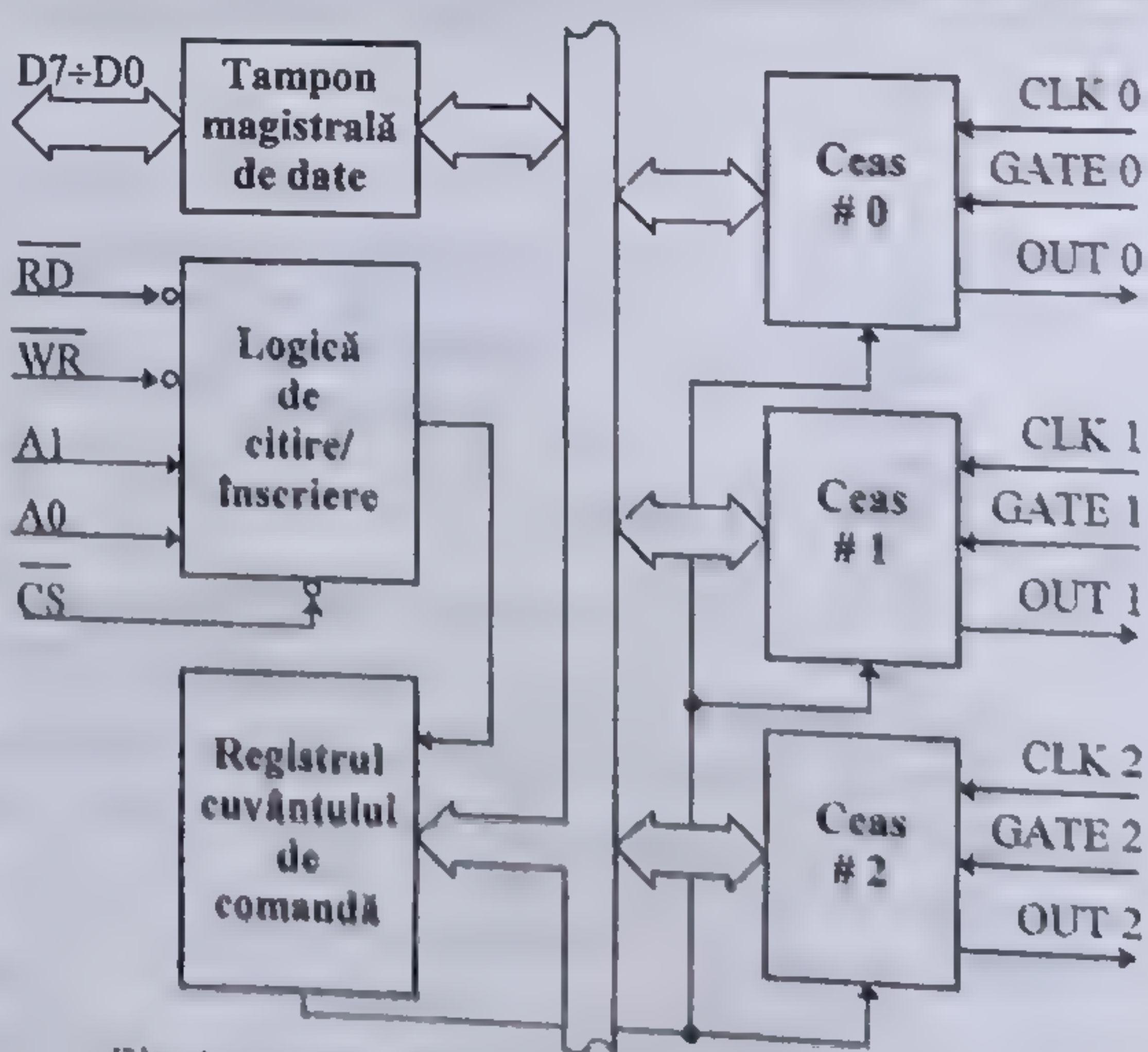


Fig.4.30. Schema bloc a circuitului PIT 8253

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

2,63MHz ($t_{CLK} = 380ns$), iar frecvența minimă nu este limitată, circuitul putând funcționa și în c.c. [43]. Înregistrarea evenimentelor se face la fiecare ciclu, reprezentat aici de o creștere urmată de o cădere a nivelului semnalului pe liniile CLK i .

Fiecare canal de timp poate funcționa în oricare din cele 6 moduri de lucru disponibile. Modul de lucru poate fi programat de către UCP printr-un cuvânt de comandă, urmat apoi de valoarea inițială a numărătorului; în acest scop se pot folosi instrucțiuni de scriere în port (OUT port). Conținutul numărătoarelor poate fi citit, în timpul decrementării, prin instrucțiuni de citire port (IN port). Pentru aceste proceduri, 8253 este prevăzut cu o logică de citire/înscrisere controlată din exterior, de către microprocesor, prin liniile \overline{RD} și \overline{WR} . Selecția unui dispozitiv 8253 se face prin activarea liniei \overline{CS} , iar adresarea internă a celor 3 ceasuri și a registrului de comandă (CWR - Control Word Register) se face prin liniile A_0 și A_1 . Aceste semnale se conectează la liniile cu aceleași nume de pe magistrala de adrese a sistemului. Microprocesorul schimbă informații cu PIT8253 prin intermediul tamponului pentru magistrala de date, format din 8 linii bidirecționale, cu facilitatea 3-stări ($D_7 \div D_0$). În interiorul dispozitivului, transferul informațiilor se realizează printr-o magistrală internă de 8 biți, fiind controlat de logica de citire/înscrisere. Rolul liniilor aferente acestui bloc rezultă din tabelul de funcționare tab.4.7.

Tab.4.7.

Se observă că primele patru linii din tabel corespund înscrierii celor trei ceasuri și a registrului cuvântului de comandă ($A_1 = A_0 = 1$). Următoarele trei linii corespund citirii ceasurilor, iar linia a 8-a indică faptul că CWR nu poate fi citit.

După alimentare, starea dispozitivului 8253 este nedefinită. Pentru a putea funcționa, fiecare ceas trebuie programat, așa cum se va arăta în paragraful următor. Canalele care nu se utilizează nu trebuie programate.

\overline{CS}	\overline{RD}	\overline{WR}	A_1	A_0	Efect
0	1	0	0	0	Încărcare numărător #0
0	1	0	0	1	Încărcare numărător #1
0	1	0	1	0	Încărcare numărător #2
0	1	0	1	1	Înscrisere registru de comandă
0	0	1	0	0	Citire numărător #0
0	0	1	0	1	Citire numărător #1
0	0	1	1	0	Citire numărător #2
0	0	1	1	1	Nici o operație - TS
1	x	x	x	x	8253 dezactivat - TS
0	1	1	x	x	Nici o operație - TS

4.5.1.1. Programarea dispozitivului 8253

Fiecare ceas (# i) trebuie programat individual, prin înscriserea unui cuvânt de comandă în registrul cu același nume (CWR), urmat de constanta de timp a canalului, de 8 sau 16 biți (CST8, respectiv CST16).

Cuvântul de comandă		In CWR
CST8	LOW (CST16) HIGH (CST16)	In # i

Ordinea programării celor trei ceasuri poate fi oarecare, în funcție de momentul de lansare a diferitelor taskuri de timp în sistem. Odată începută încărcarea constantei de 16 biți, aceasta trebuie

terminată înainte de a se trece la un alt canal. Formatul cuvântului de comandă este următorul:

Acesta stabilește numărătorul căruia i se adresează, tipul comenzi, modul de operare și modul de decrementare a numărătorului, astfel:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD
Select Counter		Read/Load		Mode		Code	

- primii doi biți mai semnificativi, D_7 și D_6 , permit selectarea unuia din cele 3 numărătoare:

SC1	SC2	Numărător selectat
0	0	# 0
0	1	# 1
1	0	# 2
1	1	nepermis

- biții D_5 și D_4 identifică tipul comenzii pentru numărătorul selectat cu D_7 și D_6 :

RL1	RL0	Citire / încărcare
0	0	Memorarea conținutului numărătorului
0	1	Citire / încărcare doar a LSB (la încărcarea LSB, MSB este automat forțat în 0)
1	0	Citire / încărcare doar a MSB (la încărcarea MSB, LSB este automat forțat în 0)
1	1	Citire / încărcare LSB, apoi MSB.

Citirea numărătorului unui ceas se poate face prin memorarea conținutului într-un registru special prevăzut în acest scop, fără a perturba funcționarea canalului, folosind o comandă cu $RL1=RL0=0$.

Comanda trebuie urmată de una sau două citiri a LSB sau/și MSB, în corespondență cu tipul constantei utilizate la programarea canalului. Se poate realiza și o citire directă a numărătorului, cu instrucțiuni *IN port*, fără memorare, dar în acest caz nu este garantată corectitudinea valorii citite.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	0	0	x	x	x	x

- biții D_3 , D_2 și D_1 selectează modul de lucru al canalului:

M2	M1	M0	Mod operare
0	0	0	Modul 0
0	0	1	Modul 1
X	1	0	Modul 2
X	1	1	Modul 3
1	0	0	Modul 4
1	0	1	Modul 5

- bitul D_0 permite selectarea modului de decrementare:

BCD

1 - numărare în BCD (4 decade)

0 - numărare în cod binar (numărător de 16 biți).

Pentru selectarea corectă a registrelor interne, la programarea PIT 8253 trebuie să se utilizeze combinațiile pentru A_1 și A_0 din tab.4.7.

4.5.1.2. Modulile de operare ale PIT 8253

Modul 0: numărător de evenimente (fig.4.31). După transmiterea cuvântului de comandă, ieșirea OUT trece pe nivel coborât. Rămâne la acest nivel atât pe durata încărcării numărătorului ($WR n$), cât și ulterior, până când s-a înregistrat numărul de evenimente programat. Evenimentele se aplică pe linia CLK, sub forma unor succesiuni de impulsuri de nivel TTL. După fiecare impuls aplicat pe linia CLK (creștere urmată de descreștere), numărătorul canalului se decrementează cu o unitate, dacă $GATE=1$. Numărarea poate fi inhibată în orice moment - prin $GATE=0$ - și se reia din punctul în care s-a întrerupt, după revenirea în "1" a semnalului pe linia GATE. La terminarea numărării (0000h în numărător), linia OUT revine pe nivel ridicat și poate genera, astfel, o cerere

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

de întrerupere (Interrupt on Terminal Count). În cazul în care se încarcă o nouă constantă în timpul numărării, aceasta va fi luată în considerare odată cu următorul impuls pe linia CLK.

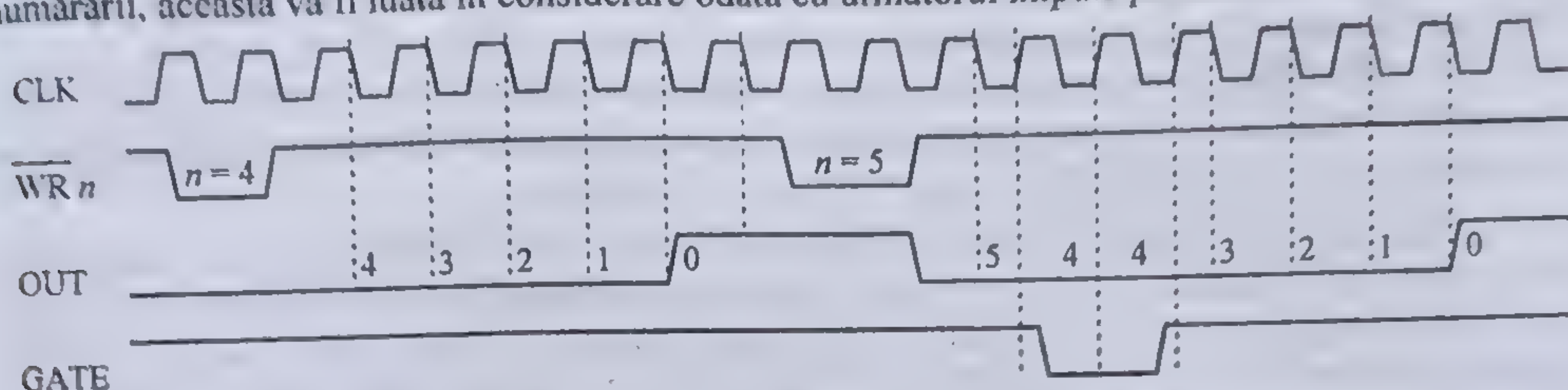


Fig.4.31. Modul 0: numărător de evenimente

Dacă pentru programarea unei noi constante se folosesc doi octeți, funcționarea canalului este afectată după cum urmează:

- înscriserea primului octet oprește numărarea imediat și OUT este forțat pe nivel logic "0";
- înscriserea celui de-al doilea octet declanșează o nouă funcționare, odată cu următorul impuls pe linia CLK.

Practic, comportarea de mai sus permite sincronizarea prin software a funcționării ceasului în modul 0.

Modul 1: monostabil programabil (fig.4.32). Ieșirea OUT va trece pe nivel coborât după o tranziție "jos-sus" a intrării GATE, odată cu următorul impuls CLK. Semnalul de declanșare aplicat pe linia GATE poate fi un impuls scurt, dar nu mai puțin de 150ns [43]. La terminarea contorizării (numărătorul canalului este vidat), linia OUT revine pe nivel ridicat. Deci, în modul 1, durata de menținere pe nivel coborât a semnalului generat pe linia OUT este *programabilă*.

O nouă funcționare, cu aceeași constantă, poate fi *redeclanșată* în timpul sau după terminarea unei temporizări, numai printr-o nouă tranziție "jos-sus" a semnalului pe linia GATE (hardware retriggerable).

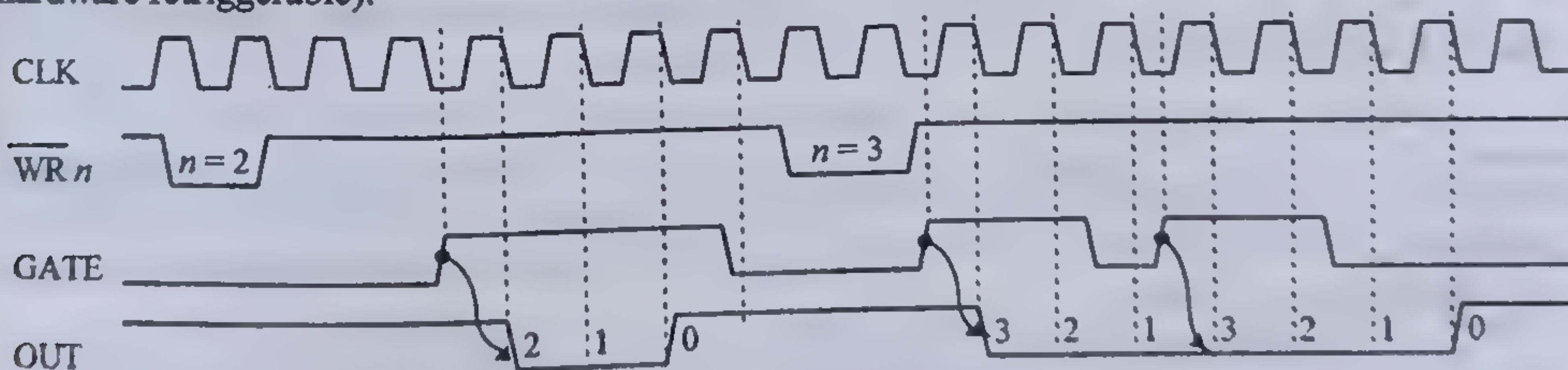
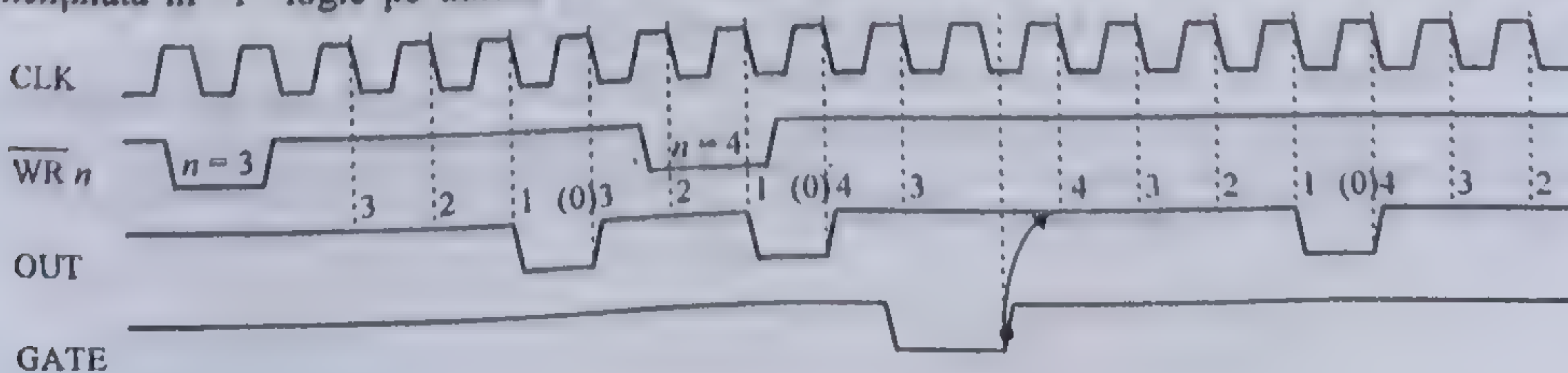


Fig.4.32. Modul 1: monostabil programabil redeclanșabil hardware

Dacă în timpul funcționării se încarcă o nouă constantă de timp, aceasta va fi luată în considerare abia după un nou front ascendent pe intrarea GATE, odată cu următorul impuls CLK.

Modul 2: generator de impulsuri (fig. 4.33). În acest mod se realizează o funcționare de tip numărător divizor prin n , unde n este valoarea cu care se încarcă contorul. Ieșirea OUT este menținută în "1" logic pe durata a $n-1$ impulsuri aplicate pe intrarea CLK și trece în "0" logic



numai pe durata celui de-al n -lea impuls. Funcționarea este periodică, fără a fi necesară reînscriserea constantei la fiecare perioadă. Dacă se înscrie o nouă constantă între două impulsuri de ieșire, perioada nu se modifică decât după trecerea prin 0 a numărătorului.

Funcționarea descrisă mai sus are loc numai dacă $GATE=1$. La o tranziție "jos-sus" a semnalului $GATE$, se redeclanșează numărarea de la valoarea înscrisă prin program, fapt ce echivalează cu o resincronizare HW a generatorului de impulsuri.

Modul 3: generator de undă dreptunghiulară (fig.4.34). Funcționarea este similară cu cea de la modul 2, cu deosebirea că la ieșire se obține un semnal de tip undă dreptunghiulară cu umplere de aproximativ 1:1. Dacă valoarea constantei înscrise la programarea ceasului (n) este pară, atunci semnalul OUT este menținut în "1" logic $n/2$ perioade ale semnalului CLK și tot $n/2$ în "0" logic. Dacă n este impar, atunci OUT se păstrează $(n+1)/2$ perioade în "1" și $(n-1)/2$ în "0" logic.

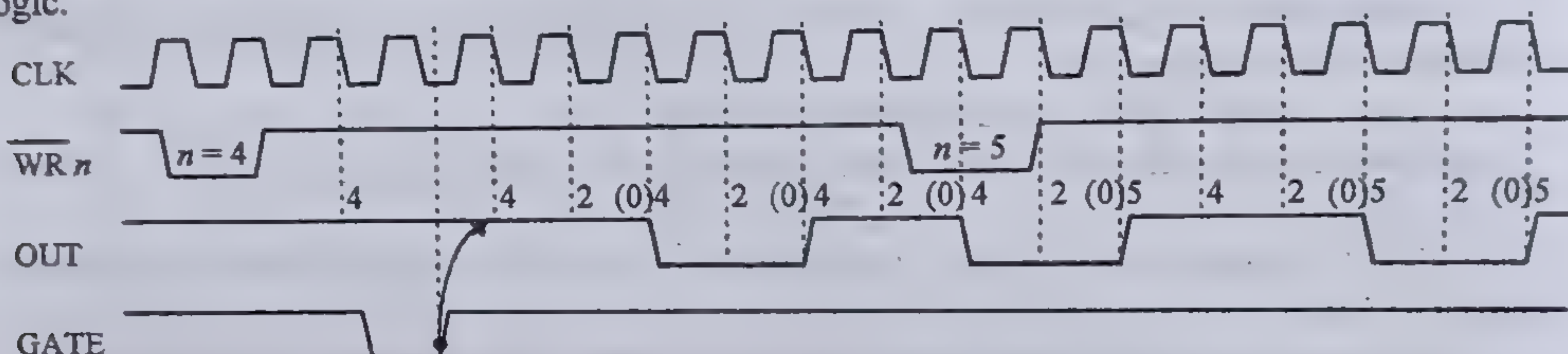


Fig.4.34. Modul 3 - generator de undă dreptunghiulară

Spre deosebire de celelalte moduri, în modul 3 numărătorul se decrementează cu 2 unități la fiecare impuls CLK , iar ieșirea OUT își comută starea la fiecare trecere prin 0000h. Excepție face cazul constantei impare când, la începutul numărării, decrementarea se face alternativ cu 1 și respectiv cu 3 unități odată, pentru a asigura un factor de umplere cât mai apropiat de 1:1.

În rest, ca și în modul 2, generatorul de undă dreptunghiulară poate fi resincronizat prin HW (la tranziția "jos-sus" a semnalului $GATE$). De asemenea, la modificarea constantei aceasta va fi luată în considerare abia la prima trecere prin 0 a numărătorului.

Modul 4: impuls (strobe) declanșat software (fig.4.35). După înscrierea cuvântului de comandă, OUT trece în "1" logic și rămâne la această valoare după ce s-a încărcat numărătorul cu constanta n , pe toată durata contorizării impulsurilor aplicate pe intrarea CLK . La terminarea temporizării, semnalul OUT trece în "0" pe durata celei de a $(n+1)$ -a perioade a semnalului CLK , după care revine în "1" logic.

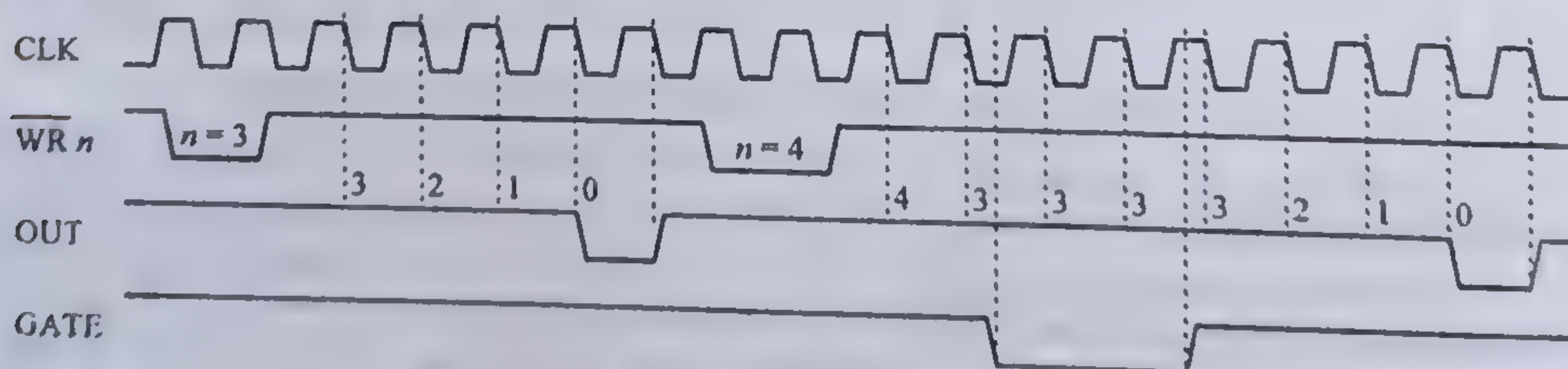


Fig.4.35. Modul 4: impuls declanșat software

Funcționarea în acest mod este reluată numai după o nouă încărcare a constantei (declanșare software). Dacă există deja o temporizare în curs, noua valoare ajunge în numărătorul cu decrementare imediat, odată cu următorul impuls CLK . Semnalul $GATE$ are și în acest caz rolul de activare/inhibare a decrementării contorului.

Modul 5: impuls (strobe) declanșat hardware (fig.4.36). Numărarea se inițiază cu frontul ascendent al semnalului GATE, fiind acompaniată de valoarea "1" a semnalului OUT. Ca și în modul 4, la terminarea contorizării OUT va trece în "0" pe durata celei de a $(n+1)$ -a perioade a semnalului CLK. Trecerea în "0" a semnalului GATE nu blochează o temporizare deja inițiată, însă un nou front crescător al acestui semnal va reinițializa funcționarea.

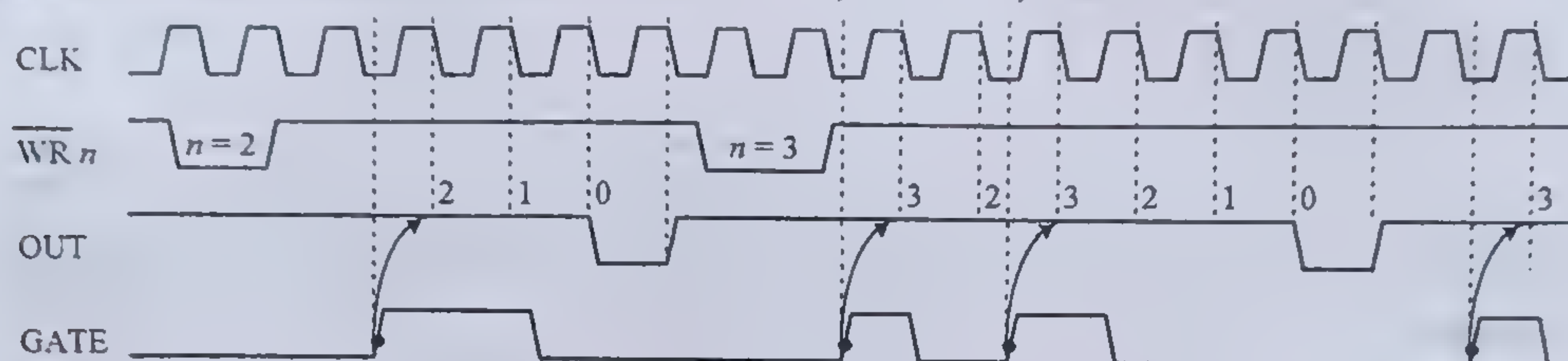


Fig.4.36. Modul 5: impuls declanșat hardware

Se poate observa că modul 5 este asemănător cu modul 1 (monostabil programabil redeclanșabil HW), cu următoarele deosebiri:

- semnalul OUT rămâne în "0" numai o perioadă a semnalului CLK;
- funcționarea unui ceas poate fi redeclanșată prin semnalul GATE, fără resetarea ieșirii OUT.

Pentru o mai bună înțelegere a modurilor de operare este util să se țină cont de efectul semnalului GATE, prezentat în tabelul următor (tab.4.8).

Tab.4.8.

Semnalul GATE Modul de lucru	Nivel coborât sau front descrescător	Front crescător	Nivel ridicat
0, 4	Inhibă numărarea	-	Autorizează numărarea
1, 5	-	1) Inițiază numărarea 2) Resetează ieșirea la următorul impuls CLK (numai în modul 1)	-
2, 3	1) Inhibă numărarea 2) Forțează ieșirea pe nivel ridicat	Inițiază numărarea	Autorizează numărarea

4.5.2. Organizarea unui SIT cu dispozitive 8253

Cu ajutorul dispozitivelor 8253 pot fi implementate sisteme de introducere a timpului (SIT) în conducerea proceselor cu microprocesoare. Fiind cunoscute taskurile de timp necesare, se pot alege modurile de operare adecvate sau o combinație a acestora, formată prin înlănțuirea canalelor de timp și adăugarea, dacă este cazul, de circuite logice suplimentare. Și în acest caz trebuie rezolvate atât aspecte HW, cât și SW.

Structura HW a unui SIT se definește ținând cont de:

- numărul și tipul taskurilor de timp;
- necesarul de dispozitive 8253 și conectarea acestora la magistrale;
- adresarea și selecția fiecărui PIT8253.

Aspectele SW implică programarea dispozitivelor 8253 în corespondență cu organizarea HW și cu tipul taskurilor.

Pentru ilustrarea modului de utilizare a circuitului 8253, considerăm următorul exemplu.

Exemplu: Într-un sistem cu 8085 ce funcționează cu $f_{XTAL}=3\text{MHz}$, să se organizeze un SIT care să asigure următoarele taskuri de timp:

- 1) ceas de timp real, în cuante de 300ms;
- 2) ceas de gardă (WDT) de 1,5s;
- 3) semnale de sincronizare pentru 2 dispozitive USART 8251, în vederea realizării a două linii de comunicație serie asincronă cu viteze de 4800, respectiv 9600baud.

Stabilirea structurii HW. Se va folosi drept semnal pilot pentru SIT ieșirea CLKOUT a microprocesorului (v.fig.4.10) care asigură un semnal TTL cu frecvența egală cu $f_{XTAL}/2 = 1,5\text{MHz}$, acceptată de PIT 8253.

Din enunțul problemei rezultă că trebuie implementate 4 taskuri de timp. Întrucât, pentru primele două taskuri, ar fi necesare valori ale constantelor de timp care depășesc capacitatea numărătoarelor, se mai introduce un task suplimentar, care să asigure un semnal de referință cu perioada de 10ms. Rezultă astfel un *necesar de 5 taskuri* de timp, ce poate fi acoperit cu două circuite 8253.

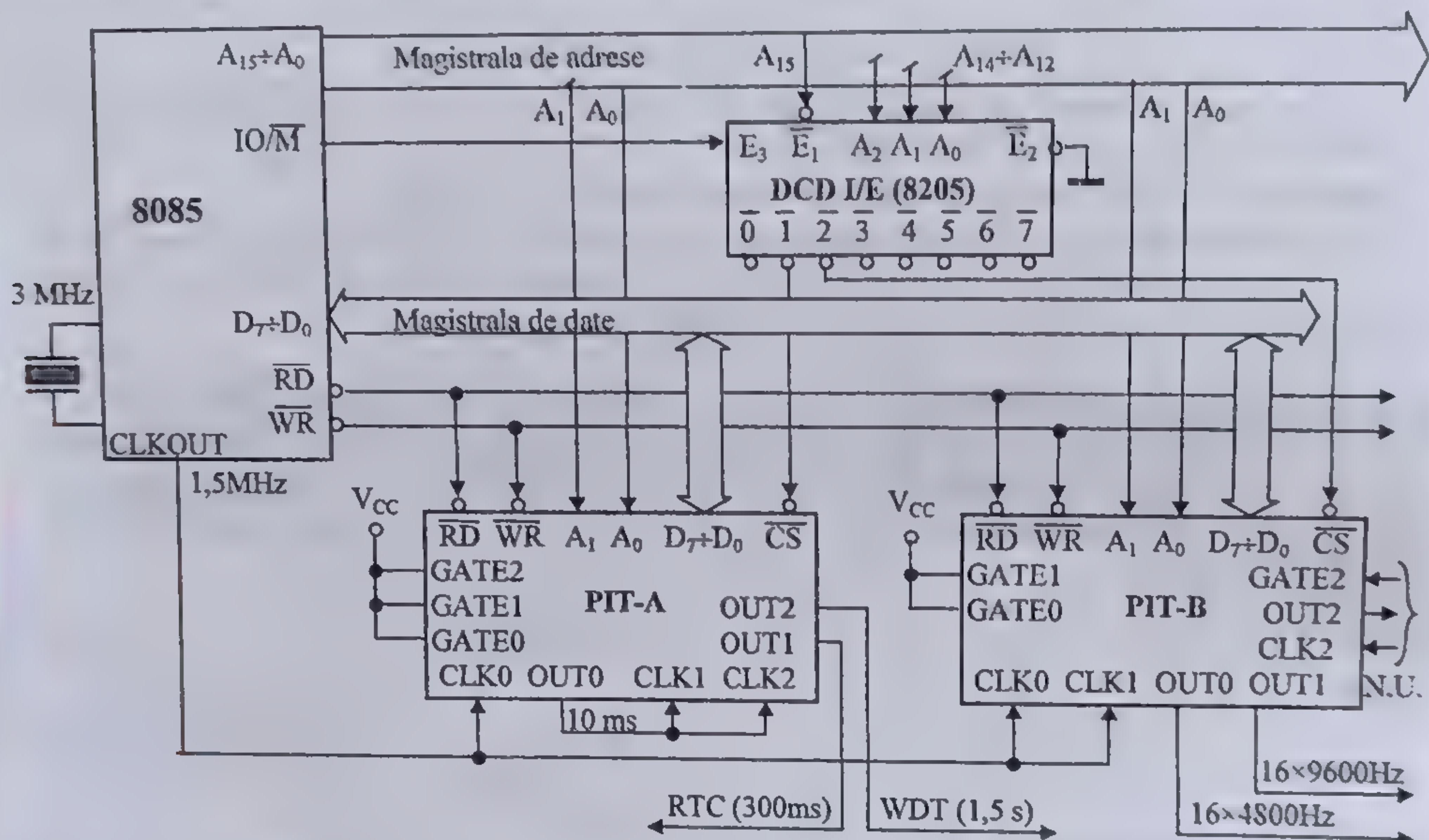


Fig.4.37. Structura hardware a SIT cu PIT 8253

Cu cele stabilite mai sus se pot defini modurile de operare pentru cele 5 ceasuri și constantele de timp aferente:

- pentru semnalul de referință de 10ms se va folosi un ceas pilotat de semnalul de 1,5MHz, care va funcționa în *modul 2*, ca divizor prin $n_1 = 1,5[\text{MHz}] \times 10[\text{ms}] = 15000$.

- ceasul de timp real, de 300ms, se va obține din semnalul de referință, la ieșirea unui ceas care va funcționa tot în *modul 2* (se poate și în *modul 3*), ca divizor prin $n_2 = 300[\text{ms}] / 10[\text{ms}] = 30$.

- ceasul de gardă va fi pilotat tot de semnalul de referință de 10ms și va fi încărcat cu constanta $n_3 = 1,5[\text{s}] / 10[\text{ms}] = 150$. O rulare normală a programului nu trebuie să depășească durata limită fixată de 1,5 s. De aceea, ceasul de gardă va fi rearmat periodic prin program, fie prin reîncărcarea constantei (*modul 4*), fie prin comanda liniei GATE (*modul 5*). Modul 4 este mai convenabil, deoarece nu necesită o logică externă suplimentară și va fi adoptat în acest caz. Dacă se folosește modul 5, se poate implementa un ceas de gardă pentru supravegherea evoluției în timp a unor evenimente externe. Un astfel de caz este, spre exemplu, supravegherea timpului de răspuns

la o solicitare de întrerupere datorată unui eveniment critic (eroare în sistem), la microprocesoarele care nu dispun de o linie de întrerupere nemascabilă, așa cum este cazul microprocesorului 8080.

- semnalul de sincronizare pentru comunicația serială de 4800 Baud trebuie să fie de formă dreptunghiulară și să aibă frecvența de $4800 \times k_v = 76800 \text{ Hz}$, unde $k_v = 16$ este factorul de viteză (k_v poate fi 1, 16 sau 64 la circuitul 8251). Rezultă necesitatea unui ceas care să funcționeze în *modul 3*, cu o constantă de timp $n_4 = 1,5 [\text{MHz}] / 76800 [\text{Hz}] = 20$.

- al doilea semnal de sincronizare, pentru comunicația de 9600 Baud, se va obține similar cazului precedent, de la un ceas funcționând tot în *modul 3*, dar cu constanta $n_5 = 10$.

Drept urmare, se poate elabora structura HW a SIT, compusă din 2 dispozitive 8253. În fig.4.37 este prezentată schema de conectare a acestor dispozitive la magistrale, cu evidențierea elementelor strict necesare.

Cele două dispozitive 8253, notate PIT-A și PIT-B, sunt selectate ca porturi I/E având adresele 10h, respectiv 20h. Primul dispozitiv 8253 (PIT-A) implementează ceasul de referință de 10ms (#0), ceasul de timp real (#1) și ceasul de gardă (#2). Cel de-al doilea dispozitiv 8253 (PIT-B) generează semnalele de sincronizare pentru cele două linii de comunicație serială de 4800, respectiv 9600Baud (#0 și #1); cel de-al treilea numărător (#2) nu este utilizat.

Programarea SIT. În corespondență cu modurile de operare și constantele de timp stabilite anterior, se realizează programarea fiecărui ceas al SIT din fig.4.37.

```

; PROGRAMARE PIT_A
PIT_A EQU 10h
CW_A_0 EQU 00110100b ; Cuvântul de comandă pentru #0: modul 2, CST16, binar.
N1 EQU 15000 ; Constanta n1 (ceas de referință de 10 ms).
CW_A_1 EQU 01010101b ; Cuvântul de comandă pentru #1: modul 2, CST8 (LSB), BCD.
N2 EQU 30h ; Constanta n2 (ceas de timp real de 300 ms).
CW_A_2 EQU 10011000b ; Cuvântul de comandă pentru #2: modul 4, CST8 (LSB), binar.
N3 EQU 150 ; Constanta n3 (ceas de gardă de 1,5 s).

; PROGRAMARE CEAS DE REFERINȚĂ
MVI A,CW_A_0 ; Se încarcă în acumulator cuvântul de comandă pentru #0.
OUT PIT_A+3 ; Se înscrie cuvântul de comandă în registrul de comandă.
MVI A,LOW(N1) ; Se încarcă LSB al constantei N1 în acumulator.
OUT PIT_A ; Se înscrie LSB pe adresa canalului 0 (PIT_A+0).
MVI A,HIGH(N1) ; Se încarcă MSB al constantei N1 în acumulator.
OUT PIT_A ; Se înscrie MSB pe adresa canalului 0.

; PROGRAMARE CEAS DE TIMP REAL
MVI A,CW_A_1 ; Se încarcă în acumulator cuvântul de comandă pentru #1.
OUT PIT_A+3 ; Se înscrie cuvântul de comandă în registrul de comandă.
MVI A,N2 ; Se încarcă constanta N2 în acumulator (CST8).
OUT PIT_A+1 ; Se înscrie constanta (LSB) pe adresa canalului 1.

; PROGRAMARE CEAS DE GARDĂ
MVI A,CW_A_2 ; Se încarcă în acumulator cuvântul de comandă pentru #2.
OUT PIT_A+3 ; Se înscrie cuvântul de comandă în registrul de comandă.
MVI A,N3 ; Se încarcă constanta N3 în acumulator (CST8).
OUT PIT_A+2 ; Se înscrie constanta (LSB) pe adresa canalului 2.

; PROGRAMARE PIT_B
PIT_B EQU 20h
CW_B_0 EQU 00010110b ; Cuvântul de comandă pentru #0: modul 3, CST8 (LSB), binar.
N4 EQU 20 ; Constanta n4 (semnal de sincronizare pentru 4800Baud).
CW_B_1 EQU 01010111b ; Cuvântul de comandă pentru #1: modul 3, CST8 (LSB), BCD.
N5 EQU 10h ; Constanta n5 (semnal de sincronizare pentru 9600Baud).

```



```

; PROGRAMARE GENERATOARE DE TACT PENTRU COMUNICAȚIA SERIALĂ
MVI A,CW_B_0      ; Se încarcă în acumulator cuvântul de comandă pentru #0.
OUT PIT_B+3        ; Se înscrie cuvântul de comandă în registrul de comandă.
MVI A,CW_B_1      ; Se încarcă în acumulator cuvântul de comandă pentru #1.
OUT PIT_B+3        ; Se înscrie cuvântul de comandă în registrul de comandă.
MVI A,N5           ; Se încarcă constanta N5 în acumulator (CST8).
OUT PIT_B+1        ; Se înscrie constanta (LSB) pe adresa canalului 1.
MVI A,N4           ; Se încarcă constanta N4 în acumulator (CST8).
OUT PIT_B          ; Se înscrie constanta (LSB) pe adresa canalului 0.

```

În secvența de programare a circuitului PIT_B s-a exemplificat posibilitatea de programare a canalelor de timp în orice ordine (de exemplu, mai întâi #1 și apoi #0); singura restricție este aceea ca încărcarea unei constante de 16 biți să se desfășoare prin două înscrieri consecutive, pe adresa aceluiași canal, fără nici o operație intermediară cu un alt ceas.

Dacă se dorește citirea la un moment dat a contorului unui canal, de exemplu al ceasului de gardă, se utilizează o procedură de citire care trebuie să respecte specificațiile de la programarea canalului în ceea ce privește numărul de octeți citați:

```

MVI A,10000000b    ; Comandă de citire "on the fly" a numărătorului canalului 2.
OUT PIT_A+3         ; Memorează conținutul numărătorului într-un registru.
IN PIT_A+2          ; Citește în acumulator conținutul registrului.

```

Trebuie menționat faptul că, în afara circuitului 8253, firma Intel mai produce un dispozitiv identic: **I8253-5**, adaptat să funcționeze cu microprocesorul 8085. De asemenea, pentru sisteme care funcționează la frecvențe mai mari, produce circuitul **I8254**, care poate lucra la o frecvență maximă de 8 și chiar 10MHz (tipul 8254-2). Toate aceste dispozitive sunt complet compatibile la pini cu 8253. În plus, dispozitivul 8254 posedă și o a treia modalitate de citire a conținutului unui numărător, printr-o procedură de citire cu memorare extinsă, numită "read-back". O astfel de citire poate fi comandată printr-un cuvânt de forma:

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
1	1	COUNT	STATUS	CNT2	CNT1	CNT0	0

Pentru specificarea procedurii "read back" se folosește combinația D₇=D₆=1, neutilizată la 8253 (v. §4.5.1.1). Spre deosebire de procedura cu memorare, când se poate citi numărătorul unui singur canal, în acest caz se poate comanda citirea tuturor numărătoarelor, printr-un singur cuvânt de comandă cu D₃=D₂=D₁=1.

Dacă bitul D₅=0 (COUNT), atunci cuvântul de comandă are același efect ca și o citire cu memorare. În schimb, dacă D₄=0 (STATUS), în registrele atașate numărătoarelor se va memora starea canalelor specificate prin biții D₃, D₂ și D₁, într-un cuvânt de stare de forma:

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
OUTPUT	NULL COUNT	RW1	RW0	M2	M1	M0	BCD

Se observă că biții D₅+D₀ conțin informațiile transmise prin cuvântul de comandă (v. §4.5.1.1). La 8254, biții D₅ și D₄ sunt denumiți Read/Write, față de Read/Load la 8253. În plus, se obțin informații și despre starea ieșirii OUT (bitul D₇), precum și dacă numărătorul canalului a fost încărcat și poate fi citit (NULL COUNT = 1).

Este posibil ca ambele opțiuni din cuvântul de comandă (COUNT și STATUS) să fie simultan active (D₅=D₄=0), întrucât starea se reține în registre separate de cele atașate

numărătoarelor. În acest caz, la prima citire va fi returnată starea, iar la următoarele se returnează LSB și/sau MSB.

Dacă s-au transmis mai multe comenzi fără a se efectua citirea, va fi reținută numai informația corespunzătoare primei comenzi.

4.6. Accesul direct la memorie în sistemele cu microprocesoare Intel de 8 biți

Datorită existenței unei singure magistrale de adrese, transferul datelor între un port de interfață al unui echipament periferic și memoria sistemului impune, în varianta clasică, intervenția microprocesorului UCP. Acesta folosește registrele sale interne ca memorie intermediară. Rezultă, în acest caz, o înseriere de dispozitive care participă la transfer, așa cum se prezintă în fig.4.38. Un astfel de transfer se efectuează în două etape: citire port sau memorie în acumulatorul microprocesorului, urmată de o înscrisiere a octetului citit în memorie, respectiv în port. De exemplu, pentru transfer din port în memorie se poate utiliza următoarea secvență de instrucțiuni:

```

CSEG ; Programul e amplasat în segmentul de cod.
nroct EQU 16 ; Se vor transfera în memorie 16 octeți,
port EQU 40h ; din portul de intrare cu adresa 40h.

; PREGĂTIREA TRANSFERULUI.
LXI H,tabmem ; Încarcă în HL adresa de memorie.
LXI B,nroct ; Încarcă în BC numărul de octeți.

; REALIZAREA TRANSFERULUI.
buda: IN port ; (10T) Transfer din port în acumulator.
MOV M,A ; (7T) Transfer din A în memorie.
INX H ; (5T) incr. pointerul de adresă.
DCX B ; (5T) Decr. contorul de octeți.
MOV A,B ; (4T)
ORA C ; (4T) Dacă B=C=0, atunci Z=1.
JNZ bucla ; (10T) Dacă mai sunt octeți, continuă.
; ..... ; altfel, încheie transferul.
DSEG ; Datele preluate din port vor fi depuse
; în segmentul de date, în RAM.
tabmem: DS nroct ; Se rezervă spațiu de memorie pentru
; cei "nroct" octeți de transferat.

```

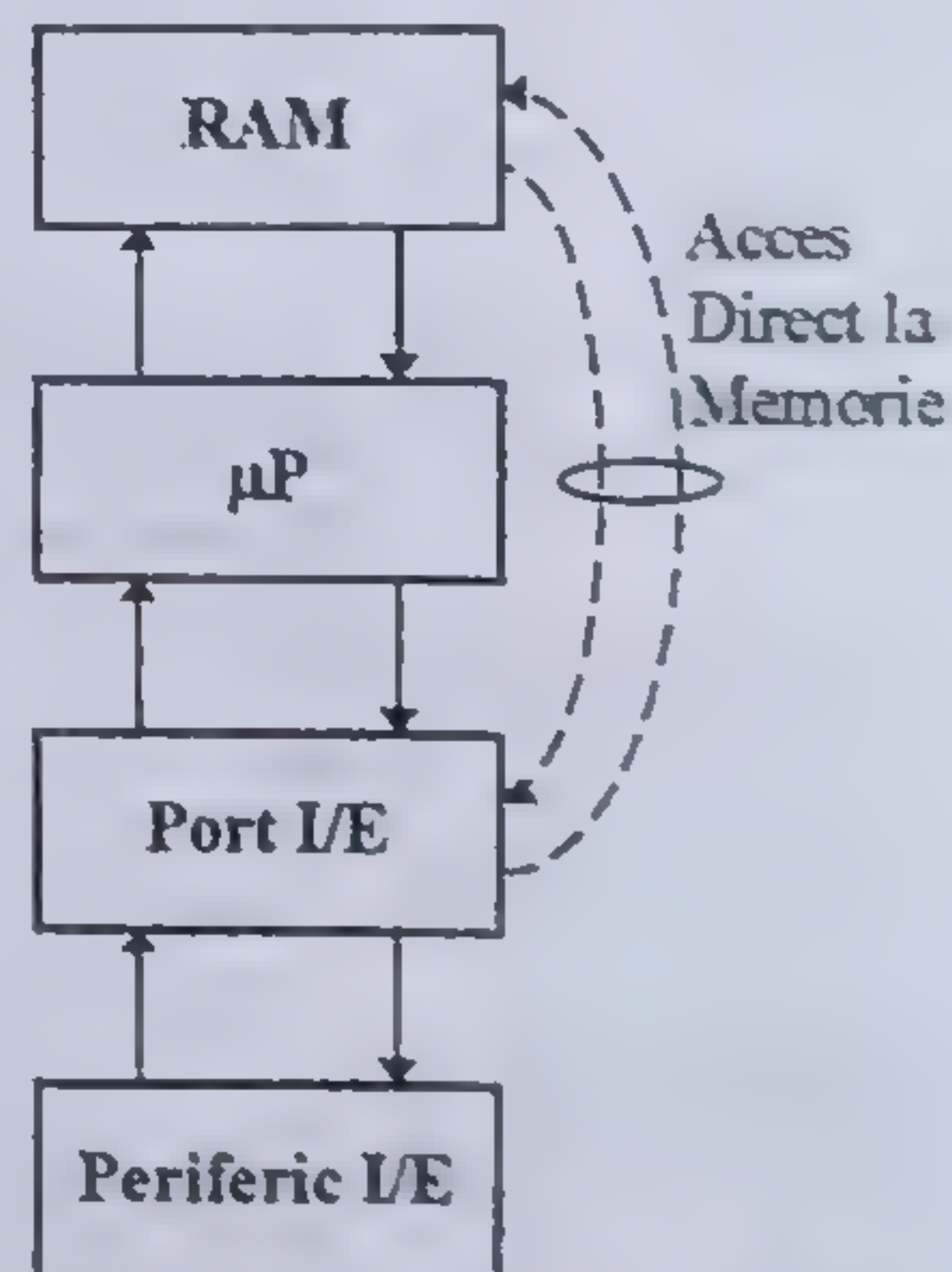


Fig.4.38. Transferul de date între memorie și un periferic I/E

Ținând cont de timpul de execuție al instrucțiunilor (v.tab.4.6), rezultă că pentru transferul unui octet sunt necesare cel puțin 45 de stări mașină (45T). Astfel, pentru un microprocesor 8080A funcționând la frecvența maximă (2MHz), se obține o rată de transfer de cca. 50Koct/s ($\approx 20\mu\text{s}/\text{octet}$). Există situații în care aceste viteze de transfer sunt insuficiente. De exemplu, memoriile externe pe disc, convertoarele A/D rapide, transmisiile de date pe fibră optică, memoriile sistemelor de afișare de tip display etc., necesită rate de transfer de ordinul sutelor de Koct/s. Soluția constă în scurtarea lanțului de dispozitive din transferul clasic, prin folosirea accesului direct la memorie. În acest caz, transferul se realizează direct între cele două dispozitive, sursă și destinație (fig.4.38), fără intervenția microprocesorului. Accesul direct la memorie este favorizat și de faptul că dispozitivele de memorie uzuale au timpi de acces $\leq 450\text{ns}$, ceea ce permite obținerea unor rate de transfer $\geq 2\text{Moct/s}$.

În sistemele cu 8080/8085, accesul direct la memorie se realizează cu ajutorul circuitelor specializate de tip *controler DMA extern* (v.§2.3.3 și fig.2.8). Acestea preiau temporar rolul

microprocesorului de coordonator al magistralei sistemului și dirijează transferurile directe port-memorie, prin intermediul semnalelor de citire/scriere generate simultan atât pentru port cât și pentru memorie. Controlerul DMA sunt orientate pe transferurile de blocuri de date, furnizând automat adresele locațiilor succesive de memorie, precum și semnalele de selecție pentru port. Firma Intel a realizat două familii de controlere DMA: I8257 și I8237, compatibile cu procesoarele de 8 și 16 biți ale firmei, dar utilizabile și cu alte procesoare (de ex. Z80).

4.6.1. Controlerul pentru acces direct la memorie I8257

Acest controler este un circuit LSI programabil (*Programmable DMA Controller*), fiind prevăzut cu patru canale DMA independente. Este împachetat într-o capsulă DIL cu 40 de pini și alimentat la o singură sursă de +5V. Este disponibil în două variante, care diferă numai prin vitezele de transfer: 8257 - o rată maximă de aproximativ 800Koct/s și 8257-5 - cu o rată maximă de 1 Moct/s. Schema bloc a acestor dispozitive este prezentată în fig.4.39.

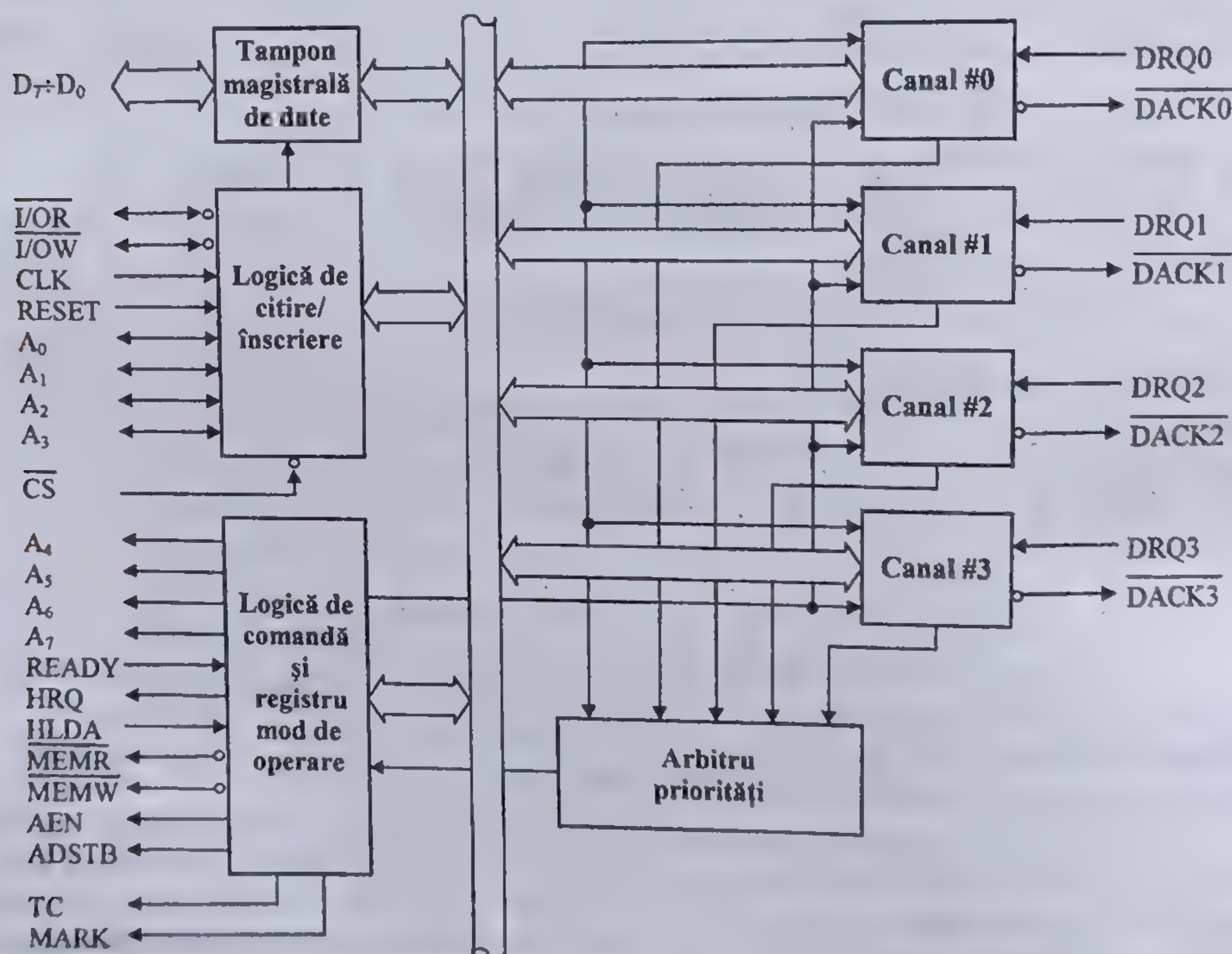
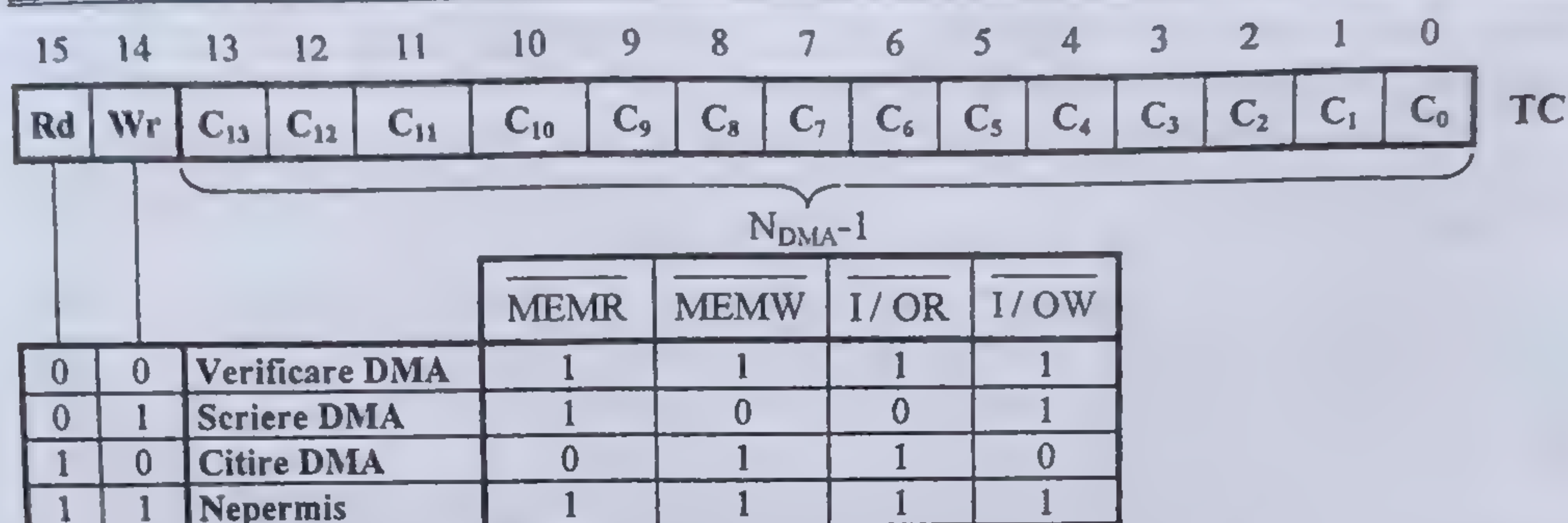


Fig.4.39. Schema bloc a circuitului DMAC 8257

Cele patru canale pentru acces direct la memorie dispun fiecare de câte două registre de 16 biți: unul al adresei DMA, iar celălalt pentru contorizarea octeților transferați (TC - Terminal Count register). Înaintea unui transfer, registrul adresei DMA trebuie inițializat cu adresa de început (cea mai mică) a zonei de memorie pentru care se solicită transferul, iar în registrul TC este programat tipul transferului DMA și numărul de octeți de transferat.



Din cei 16 biți ai registrului TC, cei mai puțin semnificativi 14 biți ($C_{13} \div C_0$) au funcția de contorizare a numărului de cicluri DMA (N_{DMA}) care mai sunt de efectuat prin canalul respectiv, adică a numărului de octeți rămași de transferat (maximum 16Koct). Pentru o funcționare corectă, datorită momentului în care este semnalizată terminarea transferului, la programarea registrului se va specifica valoarea $N_{DMA}-1$.

Cei mai semnificativi doi biți ai registrului TC condiționează generarea semnalelor de comandă pentru memorie (\overline{MEMR} , \overline{MEMW}) și portul I/E ($\overline{I/OR}$, $\overline{I/OW}$), în corespondență cu tipul transferului DMA, așa cum reiese din tabelul de mai sus.

Se observă că modul de lucru *verificare DMA* nu implică un transfer efectiv de informație. Acest mod permite verificarea corectitudinii funcționării canalului DMA, prin activarea semnalului de confirmare (\overline{DACKi}) la fiecare ciclu DMA.

De asemenea, fiecare canal dispune de câte două linii pentru dialogul cu dispozitivele externe: o linie de intrare, \overline{DRQi} (DMA ReQuest), prin care perifericul poate cere un transfer DMA și o linie de ieșire, \overline{DACKi} (DMA ACKnowledge), pentru confirmarea de către controler a acceptării solicitării de transfer.

Posibilitatea apariției unor solicitări DMA pe mai multe canale simultan a impus prezența în structura controlerului DMA a unui **arbitru de priorități** (Priority Resolver). În momentul inițializării HW, prin activarea semnalului RESET, acest bloc acordă canalului #0 prioritate maximă, iar canalului #3 prioritate minimă. Prin program se poate acorda prioritate egală (pe termen lung) celor patru canale, apelând la modul de operare cu rotirea priorităților (rotating priority). În acest caz, arbitrul de prioritate acordă sistematic prioritate minimă ultimului canal servit, iar canalului imediat următor - prioritate maximă. O dată acceptat un transfer DMA pe unul din canale, solicitările ulterioare nu întrerup transferul, chiar dacă apar pe canale cu prioritate superioară.

Prin intermediul blocului **tampon magistrală de date** (Data Bus Buffer) se realizează transferul informației între controler și microprocesorul UCP. Tamponul este bidirecțional, cu facilitare TS și, prin intermediul liniilor $D_7 \div D_0$, conectează controlerul la magistrala de date a sistemului.

În lipsa unei solicitări de transfer DMA, controlerul se află în raport de subordonare (slave) față de microprocesorul UCP. În acest caz, prin tamponul de date, microprocesorul poate inițializa registrele canalelor și registrul modului de operare sau poate citi registrele canalelor și registrul de stare. La apariția unei solicitări DMA, 8257 preia controlul magistralelor sistemului, acesta devenind coordonator de sistem (DMA master). Prin tamponul de date se transmite, la începutul fiecărui ciclu DMA, octetul superior al adresei de memorie la care are loc accesul. În restul ciclului DMA, magistrala de date este eliberată, în vederea transferului propriu-zis de date între memorie și dispozitivul I/E selectat. Deoarece adresa DMA trebuie menținută pe magistrala

de adrese pe durata întregului ciclu DMA, informația transmisă prin liniile $D_7 \div D_0$ la începutul fiecărui ciclu DMA trebuie memorată într-un registru extern.

Logica de citire/înscrisere controlează dialogul dintre 8257 și microprocesor, precum și generarea semnalelor de comandă a dispozitivelor I/E, pe durata ciclurilor DMA. Din acest motiv, liniile $\overline{I/OR}$ și $\overline{I/OW}$ sunt bidirecționale și au facilitatea TS.

Secvențierea operațiilor interne este sincronizată cu semnalul de tact, CLK, a cărui perioadă nu poate fi mai mică de 320ns la 8257, respectiv 250ns la 8257-5 [43]. Activarea semnalului RESET forțează în zero registrul modului de operare și aduce controlerul într-o stare de inactivitate (idle), cu toate canalele dezactivate.

Acestui bloc îi sunt atașate și liniile $A_3 \div A_0$, care, în *regim de slave*, permit adresarea de către microprocesor a registrelor interne ale dispozitivului 8257. Dialogul este controlat de către microprocesor prin semnalele \overline{CS} , $\overline{I/OR}$ și $\overline{I/OW}$ (intrări). În tab.4.9 se prezintă valorile asociate liniilor $A_3 \div A_0$ pentru selectarea registrelor controlerului.

Conform tab.4.9, distincția între canale se face prin liniile A_2 și A_1 , iar între registrele unui canal prin bitul A_0 . De asemenea, se poate observa că registrul modului de operare poate fi numai înscris, iar registrul de stare numai citit. Aceste două registre sunt unice și se selectează cu $A_3=1$ și $A_2=A_1=A_0=0$.

În *regim de master DMA*, liniile $A_3 \div A_0$ conțin primii patru biți mai puțin semnificativi ai adresei curente de memorie la care are loc accesul DMA. Funcționalitatea diferită în cele două regimuri de lucru justifică faptul că liniile $A_3 \div A_0$ sunt bidirecționale. Aceste linii trebuie conectate la liniile cu același nume ale magistralei de adrese a sistemului.

Tab.4.9.

Registru	\overline{CS}	A_3	A_2	A_1	A_0	Citire		Scriere	
						$\overline{I/OR}$	$\overline{I/OW}$	$\overline{I/OR}$	$\overline{I/OW}$
Adresă DMA canal #0	0	0	0	0	0	0	1	1	0
TC canal #0	0	0	0	0	1	0	1	1	0
Adresă DMA canal #1	0	0	0	1	0	0	1	1	0
TC canal #1	0	0	0	1	1	0	1	1	0
Adresă DMA canal #2	0	0	1	0	0	0	1	1	0
TC canal #2	0	0	1	0	1	0	1	1	0
Adresă DMA canal #3	0	0	1	1	0	0	1	1	0
TC canal #3	0	0	1	1	1	0	1	1	0
Mod de operare	0	1	0	0	0	1	1	1	0
Stare	0	1	0	0	0	0	1	1	1

Restul liniilor de adresă, care formează octetul inferior al adresei DMA ($A_7 \div A_4$), sunt generate de blocul logică de comandă numai când 8257 este master. Din acest motiv, liniile $A_7 \div A_4$, care se conectează la liniile cu același nume ale magistralei de adrese, au facilitatea TS.

Logica de comandă asigură corecta funcționare a controlerului pe durata ciclurilor DMA. Prin intermediul semnalelor HRQ (Hold ReQuest) și HLDA (HoLD Acknowledge), 8257 cere, respectiv microprocesorul UCP confirmă cedarea magistralelor pe durata transferului DMA. Semnalul READY asigură funcționarea DMA și în cazul sistemelor cu memorii lente, prin introducerea de stări de așteptare (WS - Wait State). Acest bloc generează și semnalele de control pentru memorie, \overline{MEMR} și \overline{MEMW} , cu facilitatea TS, întrucât se conectează la liniile similare ale magistralei de comandă.

Logica de comandă controlează și generarea corectă a adresei DMA, prin liniile AEN (Address ENable) și ADSTB (ADdress STroBe). Semnalul AEN, generat pe întreaga durată a

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

transferului DMA, specifică faptul că pe magistrala de adrese a sistemului s-a depus adresa de memorie la care are loc accesul. De asemenea, prin AEN se pot dezactiva celelalte dispozitive I/E din sistem care ar putea perturba transferul, prin activarea eronată de semnale de selecție. Semnalul ADSTB este activat la începutul fiecărui ciclu DMA, fiind folosit pentru memorarea într-un registru extern a octetului superior al adresei de memorie, transmis de controler pe magistrala de date.

În afara semnalelor mai sus menționate, logica de comandă mai emite în exterior încă două semnale: TC, care indică încheierea unui transfer DMA (atunci când contorul din registrul TC a ajuns la zero) și MARK, care se activează după fiecare bloc de 128 de octeți transferați. Ambele semnale apar numai pe durata unei stări interne (t_{CLK}), în timpul ultimului octet transmis, respectiv după fiecare 128 de octeți transferați (v.fig.4.41).

În blocul care conține logica de comandă se află și registrele generale pentru stabilirea modului de operare (Mode Set Register) și de stare (Status Register), ambele de 8 biți.

Registrul de mod (MSR) asigură validarea individuală a funcționării canalelor DMA și programarea a patru opțiuni de operare. Acest registru poate fi numai înscris (v.tab.4.9) și este șters la resetarea controlerului, pentru a preveni activarea de transferuri DMA false. Configurația registrului de mod este prezentată mai jos:

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	
AL	TCS	EW	RP	EN3	EN2	EN1	EN0	MSR

Semnificația biților este următoarea:

EN3÷EN0 (ENable channel) - activează, pe "1" logic, canalul pe care se dorește efectuarea unui transfer DMA.

RP (Rotating Priority) - stabilește schema de priorități în servirea celor patru canale care pot solicita transfer DMA. Dacă acest bit este "0", atunci fiecărui canal i se alocă, în ordine, o *prioritate fixă*: canalului #0 i se alocă prioritatea maximă, iar canalului 3 prioritatea minimă. Valoarea "1" pentru bitul RP permite *rotirea circulară a priorităților*, cu alocarea dinamică a priorității canalelor după fiecare ciclu de transfer.

EW (Extended Write) - permite generarea anticipată a semnalelor de comandă pentru scriere DMA (\overline{MEMW} , $\overline{I/O\overline{W}}$), în vederea compensării unor eventuali timpi de acces mai lungi ai dispozitivelor destinație. În lipsa acestei opțiuni ar fi necesare stări suplimentare, de așteptare, introduse printr-o logică de control a semnalului READY. Aceasta din urmă rămâne totuși singura soluție atunci când nici opțiunea EW nu este suficientă.

TCS (Terminal Count Stop) - specifică oprirea sau continuarea transferului DMA prin canalul în lucru, după terminarea transferului numărului de octeți programat în registrul TC. Acest moment este semnalat prin poziționarea în "1" logic a liniei TC (pe durata unei perioade a semnalului CLK) și prin setarea unui bit în registrul de stare. Dacă bitul TCS=1, transferul se oprește când TC=1, iar bitul de validare corespunzător (EN_i) este resetat. Un nou transfer prin același canal poate avea loc numai după o reînscrisere a registrului de mod (cu $EN_i=1$). Dacă TCS=0, transferul DMA poate continua, la adrese succesive, cât timp dispozitivul I/E menține activată linia DRQ/. Ca urmare, în acest caz, controlul încheierii transferului cade în sarcina dispozitivului solicitant și nu a controlerului DMA.

AL (Auto Load) - determină un mod special de funcționare, valabil numai pentru canalul #2, și anume *reîncărcarea automată* a parametrilor de lucru programați pentru canalul 3, fără intervenția microprocesorului. Astfel, canalul #2 se programează pentru primul transfer, iar canalul #3 se programează cu setul de parametri pentru transferul următor, care se va desfășura tot pe canalul #2, în continuarea primului transfer. Se activează apoi numai canalul #2 ($EN_2=1$); la atingerea valorii 0000h în numărătorul TC, registrele canalului #2 sunt

încărcate automat cu informațiile din registrele corespunzătoare ale canalului #3 (adresa DMA și TC), iar transferul continuă cu noii parametri de lucru.

Obs.:

- 1) Folosind opțiunea AL, este posibil să se realizeze transferuri DMA pe blocuri de date mai mari de 16 Koct (dar numai până la 32 Koct), fără intervenția microprocesorului. Mai mult, în acest caz parametrii de lucru programați pentru canalul #2 sunt automat copiați și în registrele canalului #3. Astfel, dacă acești parametri nu sunt modificați ulterior, printr-o programare separată a canalului #3, sunt posibile operații repetate pe bloc prin programarea numai a canalului #2. Acest lucru se utilizează în cazul reîmpătrării memoriei video a dispozitivelor de tip display.
- 2) Opțiunea TCS nu are efect dacă a fost programată simultan cu opțiunea AL.
- 3) Încărcarea automată a parametrilor în canalul #2 este memorată într-un bit, accesibil din registrul de stare, având denumirea UP (UPdate). În operații înlanțuite, acest bit poate fi testat de microprocesor pentru a verifica efectuarea reîncărcării și a permite astfel înscrierea unor noi parametri în registrele canalul #3.

După cum s-a menționat deja, *registrul de stare* (SR) conține informații atât despre modul de operare AL, cât și despre terminarea transferului pe canalele care au funcționat. Acest registru poate fi numai citit și este șters prin activarea semnalului RESET. În continuare este prezentată semnificația biților octetului citit din registrul de stare:

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	
0	0	0	UP	TC3	TC2	TC1	TC0	SR

TC3÷TC0 (Terminal Count) - indică, pe "1" logic, canalele pe care s-a terminat transferul programat. Biții rămân poziționați până la prima citire a SR, când devin "0". Din acest motiv, este necesară memorarea lor după fiecare citire.

UP (Update) - indică, pe "1" logic, autoinițializarea canalului #2 cu parametrii înscriși în canalul #3. La citirea de către microprocesor a conținutului registrului SR, acest bit nu se șterge. Bitul UP va fi resetat numai la programarea unui nou transfer DMA de tip "autoload".

4.6.1.1. Realizarea transferului DMA cu 8257

Un transfer DMA se desfășoară în conformitate cu graful de tranziții din fig.4.40 și diagramele de timp din fig.4.41 [43, 27]. După resetare, controlerul intră într-o stare de inactivitate, SI (Idle), ce caracterizează regimul de "slave". În această stare, microprocesorul poate programa transferuri DMA pe unul sau mai multe canale, iar controlerul DMA așteaptă apariția solicitărilor de transfer pe liniile DRQ_i (i=0÷3). La primirea unei cereri de transfer DMA, circuitul 8257 își activează linia HRQ, lansând către microprocesor un semnal de cedare a magistralelor sistemului (pe linia HOLD la 8080/8085). Ca urmare, controlerul DMA trece în starea S₀, în care așteaptă confirmarea acceptării acestei solicitări din partea microprocesorului (HLDA=1). Între timp, dacă există solicitări pe mai multe canale, se rezolvă și prioritatea în servire, fiind selectat canalul ce urmează a fi tratat.

Primirea confirmării determină trecerea controlerului în regim de "master" DMA, în starea S₁. Dacă stările SI și S₀, specifice regimului de "slave", pot fi formate dintr-un număr oarecare de perioade ale semnalului CLK, următoarele stări, notate cu S₁÷S₄, au o durată egală cu o singură perioadă a semnalului de tact. Acestea sunt specifice regimului de "master" și formează un ciclu DMA. O notă aparte face starea SW; aceasta poate interveni între S₃ și S₄ numai în anumite condiții și poate dura cel puțin o perioadă de tact.

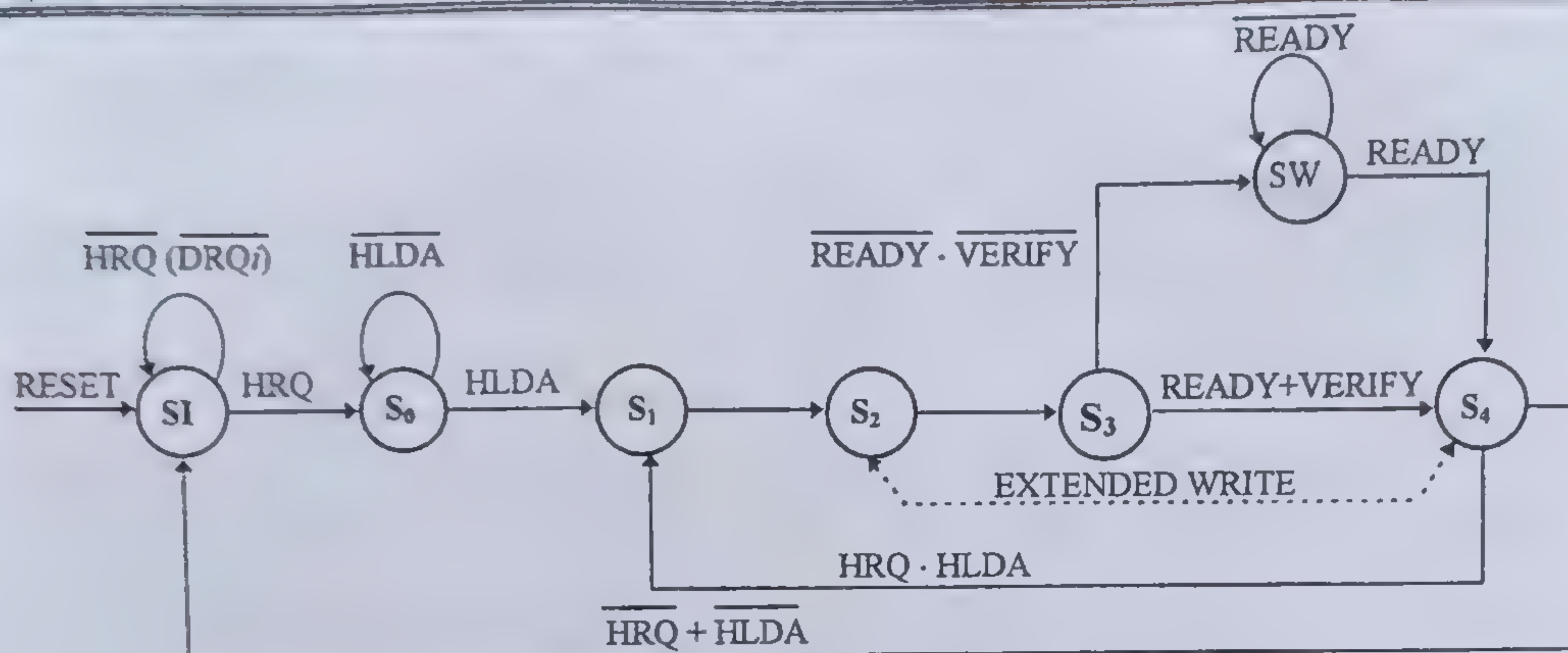


Fig.4.40. Graful de tranziții pentru circuitul 8257

După părăsirea stării S_0 se activează semnalul AEN (v.fig.4.41), care anunță faptul că 8257 va depune pe magistrala de date adresa DMA. În acest moment, toate dispozitivele I/E din sistem care nu participă la transfer trebuie dezactivate. Pe frontul pozitiv al stării S_1 , controlerul depune octetul inferior al adresei DMA curente pe liniile $A_7 \div A_0$, iar octetul superior pe liniile $D_7 \div D_0$, simultan cu activarea semnalului ADSTB. Tot în starea S_1 , dar pe frontul descrescător, i se confirmă canalului de prioritate maximă (i) acceptarea transferului. În acest moment canalul respectiv activează semnalul $\overline{DACK_i}$ și selectează astfel dispozitivul solicitant.

Apoi, 8257 trece în starea S_2 , în care începe transferul propriu-zis, prin activarea semnalului de citire și, eventual, a celui de scriere anticipată (extended write). Altfel, abia în starea S_3 se activează semnalul de scriere normală și se testează apoi starea liniei READY.

Dacă $READY=1$ sau s-a programat o operație de verificare (VERIFY), se trece în starea S_3 . În cazul în care dispozitivele externe sunt lente și se utilizează o logică de control a semnalului READY, la început $READY=0$ și atunci controlerul inserează stări de așteptare, SW, până când detectează $READY=1$. Tot în starea S_3 , dacă este cazul, se activează TC (eventual și MARK), până în S_4 .

Starea S_4 încheie un ciclu DMA cu dezactivarea semnalelor de citire/scriere, precum și a semnalului $\overline{DACK_i}$. În această stare se testează validitatea condițiilor de continuare a transferului. În caz afirmativ ($HRQ \cdot HLDA=1$), se trece la următorul ciclu DMA, începând cu starea S_1 , în care blocul cu logica de comandă incrementează adresa de memorie. Se continuă apoi cu S_2, S_3, S_4, S_1 ș.a.m.d., până când se transferă întregul bloc de octeți (burst mode). Dacă dispozitivul I/E întrerupe solicitarea ($DRQ_i=0$), ceea ce atrage $HRQ=0$, sau dacă microprocesorul părăsește regimul HOLD ($HLDA=0$), atunci 8257 trece din regimul de master DMA în starea S_1 , specifică regimului slave. La intrarea în această stare se dezactivează și semnalul AEN, fapt ce permite microprocesorului să preia controlul deplin asupra resurselor sistemului.

În cazul regimului de operare cu $TCS=1$, la activarea semnalului TC se șterge bitul de validare a canalului, din registrul de mod (EN_i). Dacă nu mai există solicitări pe liniile DRQ_i ($i=0 \div 3$), se trece în starea S_1 . În cazul în care există alte cereri în așteptare, tot în S_4 se arbitrează prioritățile și se păstrează HRQ activat. Dacă și $HLDA$ se menține activ, se continuă cu un nou transfer în modul burst, pe canalul de prioritate maximă, fără intervenția microprocesorului.

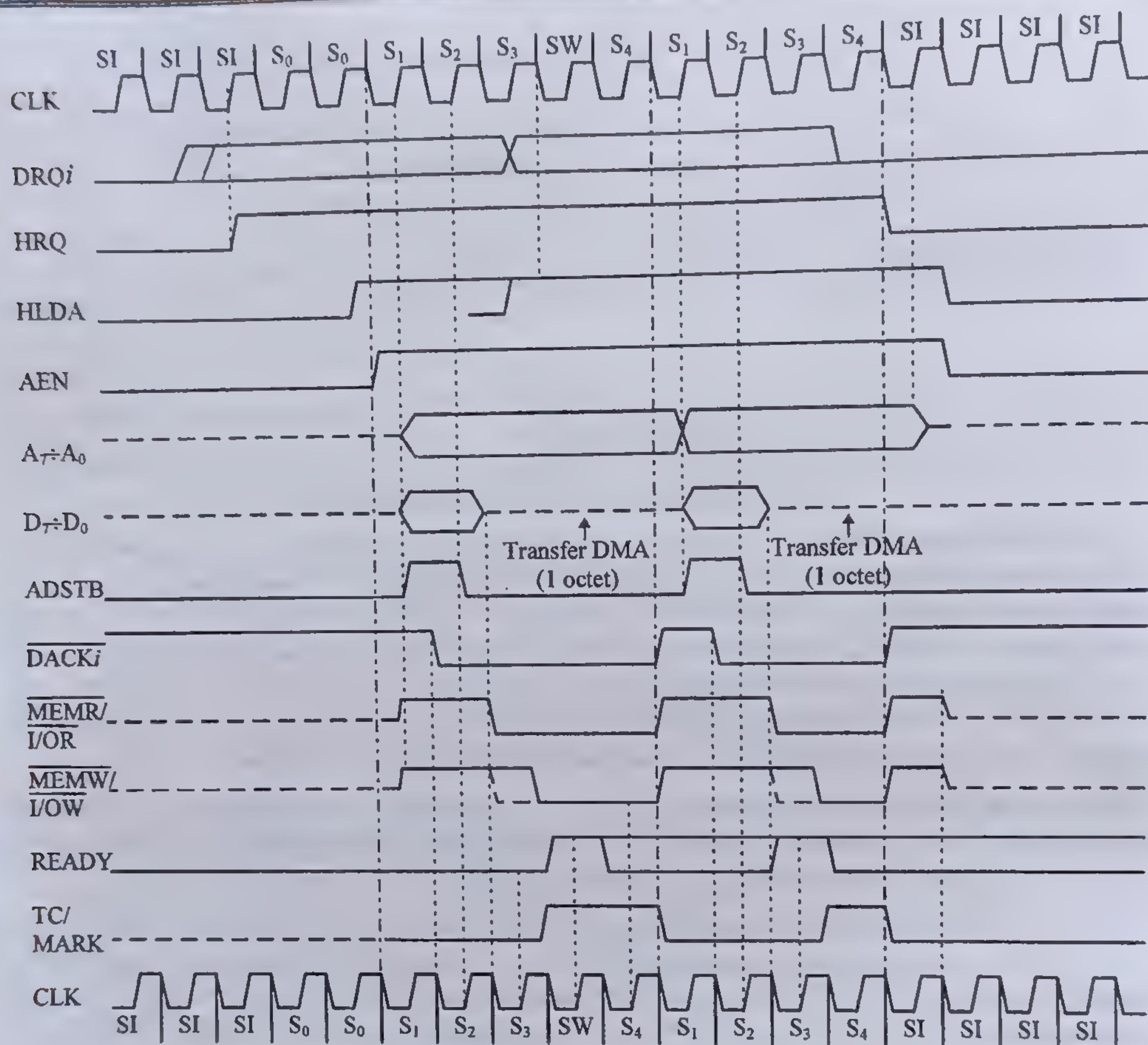


Fig.4.41. Diagrame de timp ale circuitului 8257

4.6.1.2. Programarea circuitului 8257

Fiecare transfer trebuie pregătit dinainte, prin programarea canalului DMA corespunzător. Acest lucru se realizează prin inițializarea registrelor de adresă DMA și TC ale canalului, urmată de înscrierea unui cuvânt în registrul modului de operare. Datorită faptului că aceste informații sunt transmise de către microprocesor pe liniile D_7-D_0 , registrele de 16 biți ale canalelor se programează prin înscrierea, succesivă, a doi octeți: mai întâi cel mai puțin semnificativ și apoi cel mai semnificativ. Pentru interpretarea corectă a acestora, fiecare canal are un bistabil F/L (First/Last flip-flop), care comandă logica internă de încărcare a informației în aceste registre.

Astfel, pentru programarea completă a unui transfer DMA, canalului corespunzător trebuie să i se transmită o secvență de 5 octeți, având semnificația prezentată în tab.4.10. La întocmirea acestuia s-a ținut cont de tab.4.9 și de structura registrului modului de operare, prezentate anterior. Se observă că, odată începută programarea unui registru de 16 biți, aceasta trebuie terminată (însciere LSB și MSB) înainte de a trece la următorul registru, pentru a se evita interpretarea eronată a datelor de către logica de citire/scriere a controlerului. Registrul modului de

operare se programează întotdeauna ultimul, pentru a se evita validarea canalului ($EN_i=1$) înainte de precizarea tuturor parametrilor de lucru.

Tab.4.10.

Registru	Byte	Adresa de selecție				F/L	Conținutul cuvântului							
		A ₃	A ₂	A ₁	A ₀		D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
Adresa DMA canal #i	LSB	0	x	x	0	0	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀
	MSB	0	x	x	0	1	A ₁₅	A ₁₄	A ₁₃	A ₁₂	A ₁₁	A ₁₀	A ₉	A ₈
TC canal #i	LSB	0	x	x	1	0	C ₇	C ₆	C ₅	C ₄	C ₃	C ₂	C ₁	C ₀
	MSB	0	x	x	1	1	Rd	Wr	C ₁₃	C ₁₂	C ₁₁	C ₁₀	C ₉	C ₈
Mod operare	-	1	0	0	0	-	AL	TCS	EW	RP	EN3	EN2	EN1	EN0

Exemplu: Să se programeze transferul unui bloc de date de 1Koct, prin canalul #1 al unui circuit 8257, în memoria sistemului, începând cu adresa 0100h. Se cunoaște că circuitul 8257 este selectat ca dispozitiv I/E cu adresa de bază 50h, iar dispozitivele de memorie sunt rapide.

Din specificațiile de mai sus rezultă că prin canalul #1 se va desfășura o procedură de scriere DMA în memoria RAM a sistemului, fără utilizarea de stări de așteptare. Ținând cont de informațiile din tab.4.10, pregătirea transferului DMA necesită execuția următoarei secvențe de program:

```
; Se va transfera 1Koct de date pe canalul #1, de la un dispozitiv de intrare în memorie, la 0100h.
DSEG          ; Segmentul de date care va fi amplasat în memoria RAM.
```

```
nroct EQU 1024 ; 1 Koct = 1×210 = 1024 octeți
ORG 0100h ; Stabilește adresa de amplasare a datelor (0100h).
bufmem: ; (bufmem=0100h)
DS nroct ; Rezervă spațiu pentru cei 1024 de octeți care vor fi transferați.
```

```
CSEG          ; Secvența următoare de program face parte din segmentul de cod.
```

```
; PREGĂTIREA TRANSFERULUI DMA PRIN CANALUL #1
```

```
MW EQU 01100010b ; Cuvântul de mod de operare, cu TCS=EW=EN1=1.
DMAC EQU 50h ; Adresa de bază a controlerului DMA este 50h.
CHAN EQU 1 ; Se va utiliza canalul #1.
```

```
; Programarea adresei DMA (bufmem=0100h):
```

```
MVI A,LOW(bufmem) ; mai întâi LSB,
OUT DMAC+CHAN*2 ; (pe adresa registrului DMA al canalului #1, aici 52h)
MVI A,HIGH(bufmem) ; apoi MSB,
OUT DMAC+CHAN*2 ; pe aceeași adresă.
```

```
; Programarea registrului TC:
```

```
MVI A,LOW(nroct-1) ; mai întâi LSB,
OUT DMAC+CHAN*2+1 ; (pe adresa registrului TC al canalului #1, aici 53h)
MVI A,HIGH(nroct-1) OR 01000000b ; apoi MSB (cu Rd=0, Wr=1)
OUT DMAC+CHAN*2+1 ; pe aceeași adresă.
```

```
; Programarea registrului modului de operare:
```

```
MVI A,MW ; înscrie octetul MW definit mai sus
OUT DMAC+8 ; pe adresa registrului MSR al circuitului, aici 58h
```


; În acest moment, canalul #1 este complet programat și microprocesorul trebuie să anunțe dispozitivul de intrare că poate solicita pe linia DRQ1, în orice moment, efectuarea transferului DMA.
 ; Transferul DMA se execută la cererea dispozitivului de intrare.
 ; Microprocesorul poate determina sfârșitul transferului DMA, testând într-o buclă bitul TC1 din registrul de stare:

buclo:	IN	DMAC+8	; Citește registrul de stare (SR, aici având adresa 58h).
	ANI	00000010b	; Z=1 dacă și numai dacă TC1=0.
	JZ	buclo	; Salt la adresa "buclo" dacă transferul nu s-a încheiat.

4.6.1.3. Organizarea accesului direct la memorie cu circuitul 8257

Din cele prezentate anterior rezultă că, folosind un controler 8257, se poate organiza un sistem cu patru canale de acces direct la memorie. Structura HW a unui astfel de sistem implică folosirea unui registru (latch) extern pentru reținerea octetului superior al adresei DMA, transmis multiplexat în timp de 8257 pe magistrala de date. De obicei se folosește un circuit 8212, cu funcționare în regimurile DATA IN / DATA LATCH (v.fig.1.25 în §1.1.3.2), controlat de 8257 prin semnalele AEN și ADSTB. Pot fi utilizate și alte dispozitive, cum ar fi circuitul 74LS373, cu mici adaptări pentru semnalele de comandă.

În fig.4.42 este prezentată conectarea circuitului 8257 într-un sistem cu 8080A. Dacă microprocesorul funcționează la frecvența de 2MHz, rata maximă de transfer DMA este de 500Koct/s. Aceasta se poate calcula știind că pentru transferul unui octet sunt necesare patru perioade ale semnalului CLK, care este conectat la ceasul sistemului, Φ_{TTL} .

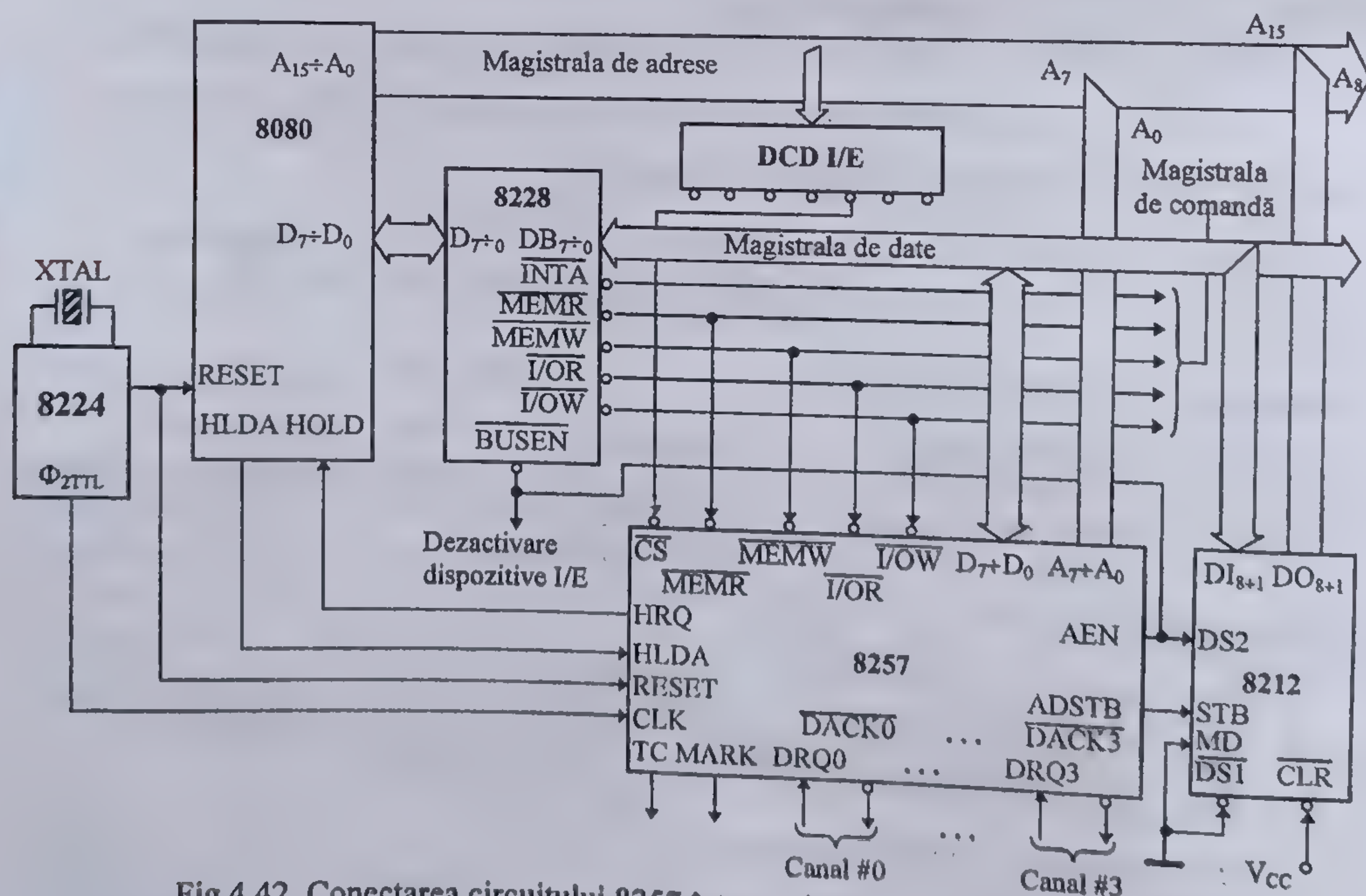


Fig.4.42. Conectarea circuitului 8257 într-un sistem cu microprocesor 8080

În cazul în care sistemul este realizat cu microprocesorul 8085A, schema de conectare a controlerului 8257 este prezentată în fig.4.43. Pentru decodificarea semnalelor de control se utilizează un circuit SN 74LS257, care este un multiplexor dublu de două căi. Căile sunt controlate

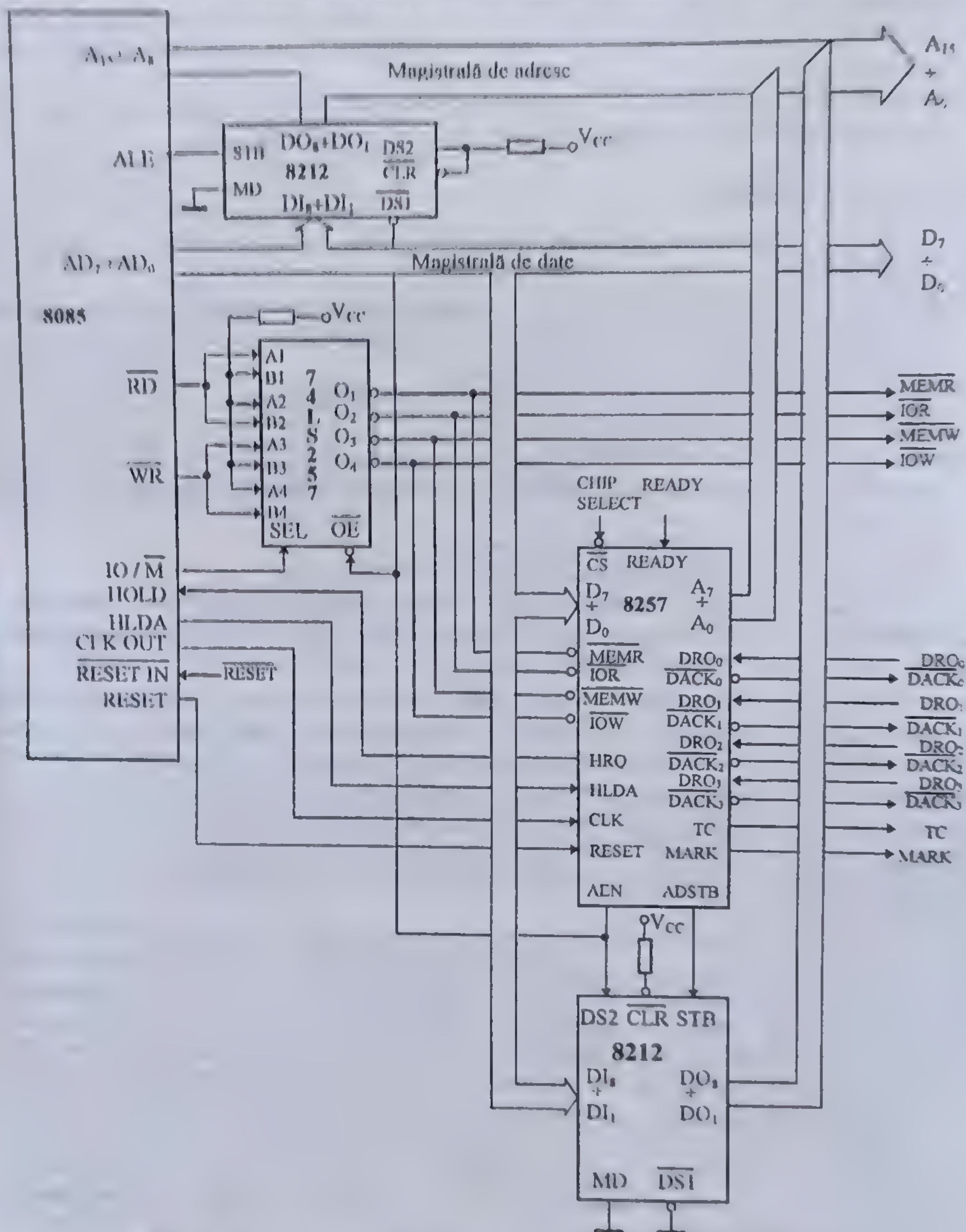


Fig.4.43. Schema de conectare a DMAC 8257 într-un sistem cu 8085A de semnalul $\overline{IO/M}$, prin intermediul liniei SEL. Astfel, se generează aceleași semnale ca și în cazul folosirii circuitului 8205 (v.fig.4.15).

Deși aceste sisteme de acces direct la memorie sunt suficient de puternice pentru majoritatea aplicațiilor, pot fi semnalate următoarele limitări:

- dificultatea extinderii la mai mult de patru canale DMA;
- necesitatea unor dispozitive suplimentare pentru realizarea unui transfer memorie-memorie;
- limitarea la 16Koct a dimensiunii blocurilor transferate pe un canal, în particular până la 32Koct prin canalul #2;
- după terminarea unui transfer, canalele nu pot fi reinițializate automat, fără intervenția microprocesorului, cu excepția canalului #2 în modul cu autoîncărcare.

4.6.2. Controlerul 8237A

Circuitul 8237A este o variantă îmbunătățită a controlerului 8257 și are următoarele caracteristici principale [43, 37]:

- conține 4 canale DMA independente, cu posibilitatea de autoinițializare după terminarea transferului unui bloc, lucrând cu incrementarea/decrementarea adresei de memorie;
- poate realiza transferuri memorie-memorie;
- viteză maximă de transfer de până la 1,6 Mcuv/s (pentru 8237A-5);
- este expandabil pentru orice număr de canale;
- poate masca individual solicitările de acces la cele 4 canale;
- nivelul activ al semnalelor de interfață cu perifericele, $DREQ_i$ și $DACK_i$ ($i=0\div3$), este programabil;
- poate executa transferuri DMA inițiate software;
- dimensiunea maximă a unui bloc transferat este de 64 Kcuvinte.

În afară de circuitul standard (8237A) care funcționează, ca și 8257, la o frecvență de max. 3MHz ($t_{CLK} = 320ns$), există și o variantă pentru 4MHz ($t_{CLK}=250 ns$) - 8237A-4, precum și una pentru 5MHz ($t_{CLK} = 200ns$) - 8237A-5 [43].

Schema bloc simplificată a controlerului 8237A, din fig.4.44, ilustrează în principal conexiunile de date și mai puțin diversitatea semnalelor de control dintre blocuri. Spre deosebire de familia 8257, în acest caz fiecare din cele 4 canale DMA cuprinde: două *registre de adresă* (de bază și curentă), de 16 biți, două *contoare de cuvinte* (de bază și curent), tot de 16 biți, precum și un *registru al modului de operare*, de 6 biți. În *registrele de bază* se specifică adresa locației de memorie de unde începe transferul și respectiv numărul de cicluri DMA. În timpul transferului, conținutul registrelor de bază rămâne neschimbat, permițând realizarea regimului de *autoinițializare*.

La programare, conținutul registrelor de bază se încarcă automat în *registrele curente*, care se modifică în timpul transferului DMA: registrul curent de adresă actualizează adresa memoriei prin incrementare sau decrementare (la alegere), iar contorul curent se decrementează. *Decrementorul și incrementorul/decrementor*, împreună cu registrele temporare ale contorului de cuvinte și adresei DMA, asigură actualizarea adresei curente și a numărului de cuvinte rămase de transferat. Registrele de bază pot fi numai înscrise, în timp ce registrele curente pot fi și citite.

Registrul modului de operare este de fapt un registru de 8 biți, din care numai 6 biți sunt utilizați pentru stabilirea modului de operare pentru canalul respectiv.

Ca și la 8257, blocul cu *logica de comandă* asigură secvențierea operațiilor interne și generarea comenzilor externe. De asemenea, există și un *bloc de codificare a priorităților* și de stabilire a tipului de priorități folosite: *fixe* sau *rotitoare*. Funcționalitatea liniilor externe este aceeași cu a semnalelor cu același nume de la 8257, cu excepția semnalului \overline{EOP} (END OF PROCESS).

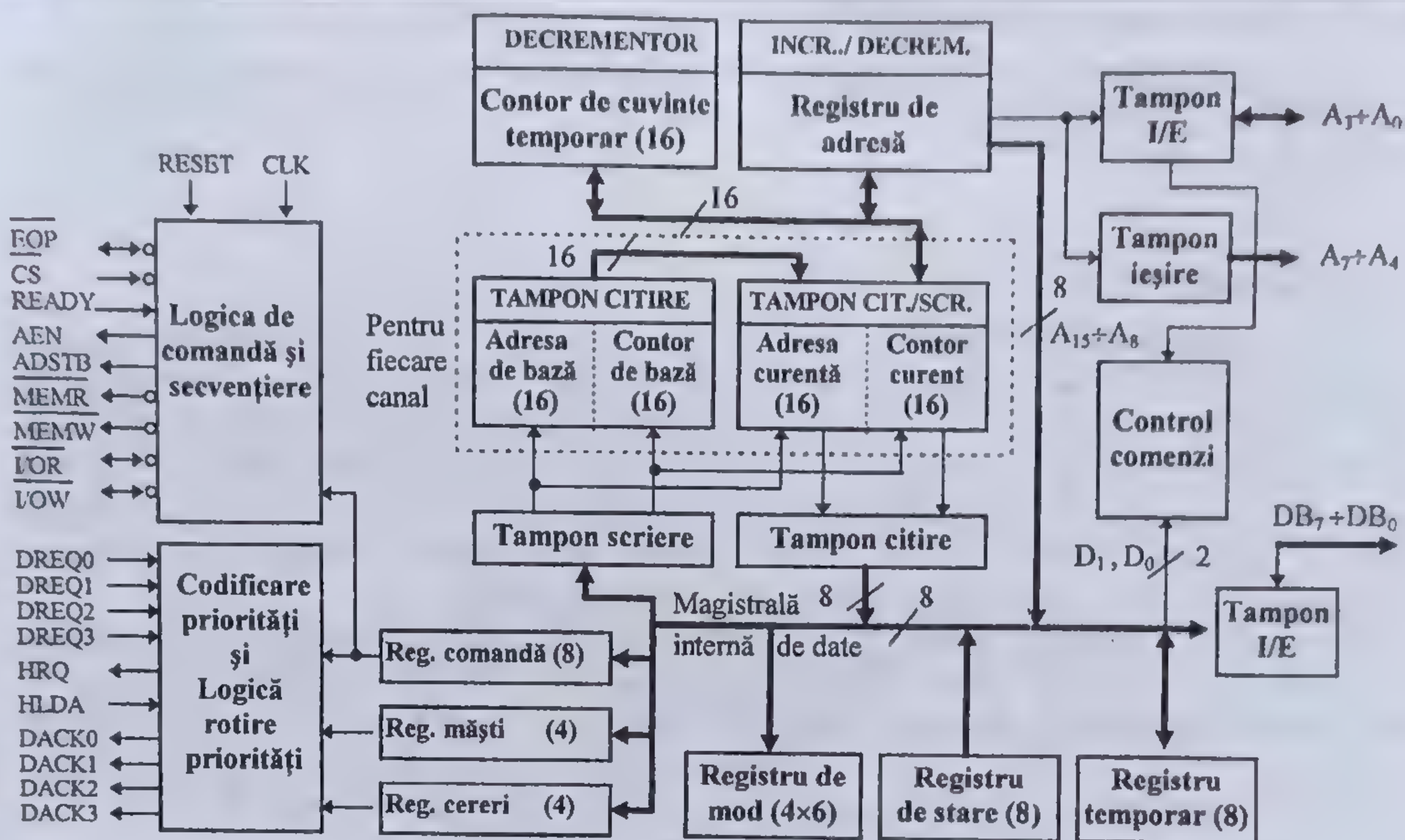


Fig.4.44. Schema bloc a circuitului 8237A

Aceasta este o linie bidirecțională, activă pe nivel coborât și îndeplinește, *ca ieșire*, rolul semnalului TC de la 8257: la terminarea transferului printr-un canal se generează un impuls pe linia \overline{EOP} . *Ca intrare*, \overline{EOP} forțează încetarea transferului pe canalul activ, resetarea cererii, setarea bitului TC în registrul de stare și mascarea canalului respectiv. În plus, dacă este programată autoinițializarea, la o comandă \overline{EOP} se realizează copierea informației din registrele de bază în registrele curente, dar nu se maschează canalul.

Trebuie menționat faptul că la încheierea transferului DMA printr-un canal se generează și un semnal *EOP intern*, având același efect ca și în cazul în care acesta este activat din exterior. În situația în care serviciul \overline{EOP} extern nu se utilizează, se recomandă ca linia respectivă să fie conectată printr-un rezistor la V_{CC} , pentru a preveni întreruperea accidentală a unui transfer DMA.

Blocul pentru controlul comenzilor asigură interpretarea comenzilor de la microprocesor. Similar blocului cu logica de citire/înscrisere de la 8257, acesta controlează procedurile de citire/înscrisere a registrelor circuitului 8237A în regimul de "slave". Registrele interne ale controlerului 8237A formează o memorie internă de 344 biți și sunt prezentate în tab.4.11.

O descriere detaliată a registrelor, precum și funcționalitatea acestora, vor fi prezentate ulterior.

Tab.4.11.

Denumire			Dimensiune	Nr.
Registru	de adresă	de bază	16 biți	4
		curentă	16 biți	4
		temporar	16 biți	1
	contor	de bază	16 biți	4
		curent	16 biți	4
		temporar	16 biți	1
	de mod de operare		8 biți	4
	de stare		8 biți	1
	de cereri		4 biți	1
	de măști		4 biți	1
	temporar		8 biți	1

4.6.2.1. Moduri de transfer la 8237A

Spre deosebire de 8257, circuitul 8237A poate efectua transferuri DMA, în regim de "master", în patru moduri distincte:

- singular;
- în bloc;
- la cerere;
- în cascadă.

Modul de transfer singular (Single Transfer Mode) - în acest mod se execută transferul unui singur octet, apoi se cedează controlul UCP. Astfel, pe perioada transferului se pot intercala cicluri DMA cu cicluri mașină ale UCP. La fiecare transfer singular, contorul de cuvinte se decrementează, iar adresa se decrementează sau se incrementează. Când contorul de cuvinte realizează tranziția de la 0 la FFFFh, se generează un impuls \overline{EOP} intern, cu efectele menționate anterior. Semnalele $DREQ_i$ ($i=0\div3$) trebuie menținute active până la activarea semnalelor de confirmare $DACK_0\div3$, care indică astfel recunoașterea solicitărilor.

Modul de transfer în bloc (Block Transfer Mode) este similar modului "burst", cunoscut de la circuitul 8257. La o cerere pe linia $DREQ_i$, controlerul efectuează transferul neîntrerupt al întregului bloc programat, până când contorul de cuvinte trece de la 0 la FFFFh sau până când se activează linia externă \overline{EOP} . Ca și la modul anterior, $DREQ_i$ trebuie să fie menținut activ până la activarea semnalului de confirmare corespunzător, $DACK_i$. Dacă s-a programat autoinițializarea, atunci canalul va fi reinițializat la terminarea transferului.

Modul de transfer la cerere (Demand Transfer Mode) este asemănător cu modul anterior, cu deosebirea că terminarea transferului cade în sarcina dispozitivului extern. Acesta trebuie să dezactiveze $DREQ_i$ la epuizarea datelor. Dacă, după o dezactivare temporară a semnalului $DREQ_i$, dispozitivul I/E dorește să continue un transfer DMA, acesta poate fi reluat prin reactivarea semnalului $DREQ_i$, după acceptarea lui de către controler. În intervalul de timp în care transferul DMA a fost dezactivat, valorile intermediare ale adresei și a numărului de cuvinte rămân memorate în registrele curente ale canalului. Și în acest caz, un semnal \overline{EOP} extern încheie transferul DMA. Dacă se utilizează opțiunea de autoinițializare, numai un semnal extern \overline{EOP} poate cauza reîncărcarea, după terminarea transferului precedent.

Modul de lucru în cascadă (Cascade Mode) permite expandarea unui sistem de acces direct la memorie, prin conectarea unor circuite 8237A suplimentare la liniile $DREQ_i$ și $DACK_i$. În fig.4.45 se prezintă un astfel de sistem, cu 10 canale DMA. În mod similar, sistemul se poate extinde la 16 sau mai multe canale, prin adăugarea altor două circuite 8237A pe canalele 2 și 3 ale circuitului 8237A de pe nivelul 1, respectiv prin introducerea unui nivel suplimentar.

Un canal al circuitului de pe nivelul 1, programat să funcționeze în modul de lucru în cascadă, are numai rolul de a arbitra prioritățile și de a comunica cu microprocesorul și cu circuitele de pe nivelul 2 prin semnalele HRQ , $HLDA$, respectiv $DREQ_i$ și $DACK_i$. Liniile \overline{MEMR} , \overline{MEMW} , $\overline{I/OR}$ și $\overline{I/OW}$ ale acestui circuit sunt dezactivate, pentru a se evita apariția unor conflicte cu canalele active ale dispozitivelor de pe nivelul 2.

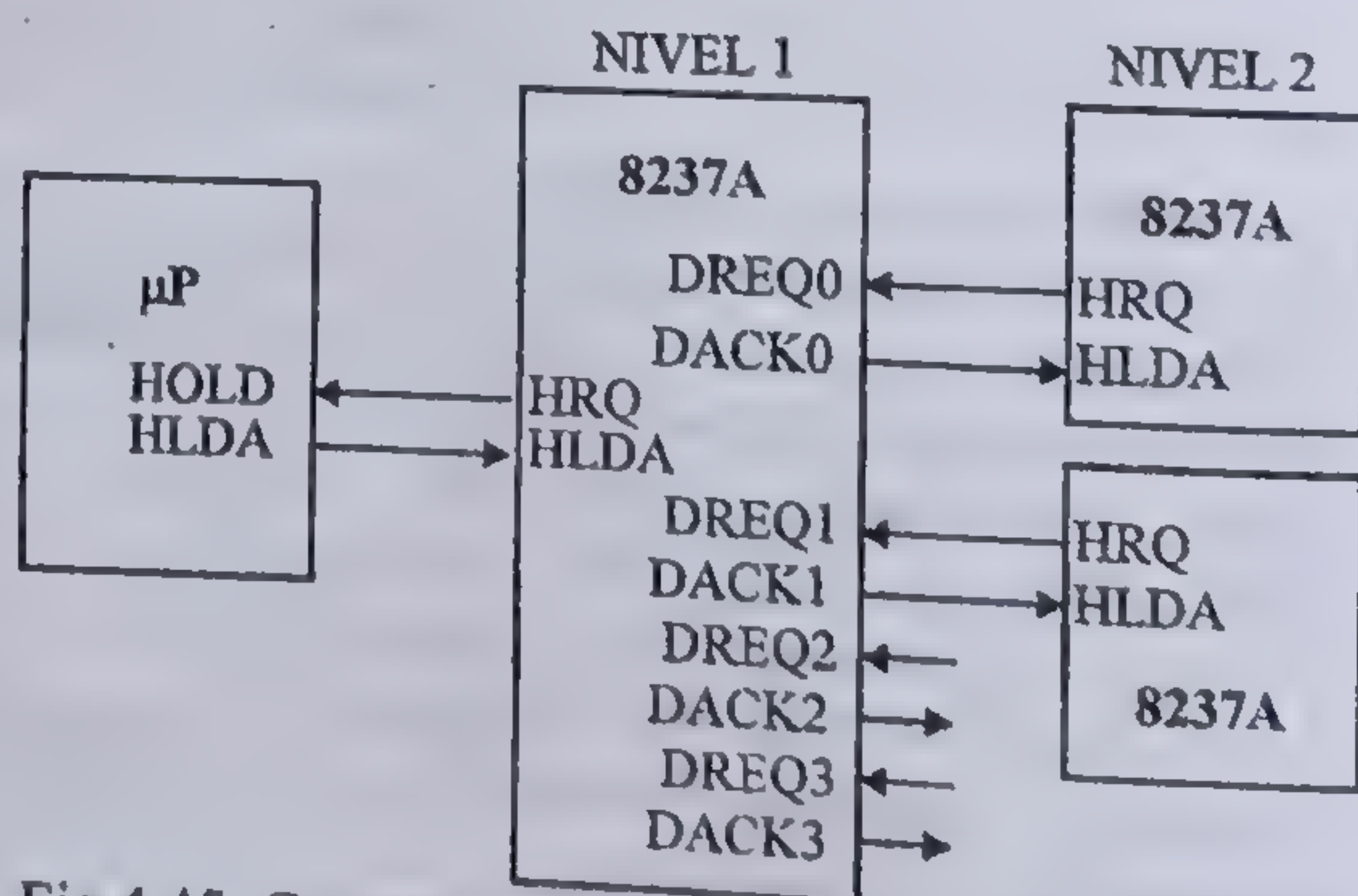


Fig.4.45. Conectarea în cascadă a circuitelor 8237A

4.6.2.2. Tipuri de transfer DMA la 8237A

Ca și la controlerele 8257, se pot realiza toate cele trei tipuri de transfer între memorie și dispozitivele I/E: *citire, scriere și verificare DMA*. În afara acestora, la 8237A apar două tipuri suplimentare:

- transferul memorie-memorie;
- transferul cu autoinițializare.

Transferul memorie-memorie (Memory to Memory) permite copierea unui bloc de date dintr-o zonă de memorie în alta. Această procedură se poate realiza numai prin intermediul canalelor #0 și #1. Canalul #0 conține *adresa sursă*, iar canalul #1 *adresa destinație*. Transferul se declanșează prin program, odată cu setarea bitului pentru canalul #0 din *registru de cerere*. Ieșirile DACK0 și DACK1 nu sunt activate. Octetul specificat prin adresa curentă din canalul #0 este adus într-un *registru temporar* și este apoi înscris la adresa destinație, specificată de canalul #1. Când contorul curent de cuvinte al canalului #1 trece din 0 în FFFFh, transferul se încheie, fapt semnalizat prin $\overline{\text{EOP}}$.

Canalul #0 poate fi programat să mențină aceeași adresă pentru toate transferurile. În acest caz, este posibilă umplerea unui spațiu de memorie (la destinație) cu același octet. De asemenea, deoarece pe durata unui transfer memorie-memorie un semnal $\overline{\text{EOP}}$ extern este acceptat, se pot concepe scheme hardware de comparare, care să detecteze un anumit octet în blocul transferat. Activarea semnalului $\overline{\text{EOP}}$ la coincidență asigură oprirea transferului DMA.

Transferul cu autoinițializare (Autoinitialize) se poate efectua prin oricare din cele patru canale, setând bitul al 4-lea în *registru de mod*. Pe durata procesului de autoinițializare, care poate avea loc după un impuls $\overline{\text{EOP}}$, conținutul registrelor de bază se copie în registrele curente ale canalului activ (#i). În acest caz, canalul poate efectua din nou un transfer DMA la detectarea unei cereri DREQi valide.

4.6.2.3. Efectuarea transferului DMA cu 8237A

Transferul între memorie și un dispozitiv I/E decurge în mod similar cu cel efectuat de circuitul 8257, în conformitate cu graful din fig.4.40 și cu diagrama de semnale din fig.4.41. Astfel, în starea SI (idle), 8237A testează liniile DREQi (i=0÷3) la fiecare ciclu de tact, pentru a determina apariția unei solicitări DMA. De asemenea, controlerul testează linia $\overline{\text{CS}}$, în așteptarea unei proceduri de înscriere sau de citire a registrelor sale interne de către microprocesor. Atunci când $\overline{\text{CS}}=0$ și HLDA=0, 8237A intră în *regim de programare* (Program Condition), permițând UCP să înscrie sau să inspecteze conținutul registrelor sale interne. În plus, controlerul acceptă și comenzi software (Software Commands) speciale, care nu utilizează magistrala de date. Programarea se poate face numai până când se recepționează un semnal HLDA=1 de la microprocesor, după care controlerul părăsește regimul slave și începe primul ciclu DMA.

Pentru transferul unui cuvânt sunt parcurse succesiv stările S₁, S₂, S₃ și S₄, cu posibilitatea inserării de stări de așteptare între S₂ (sau S₃) și S₄, dacă se utilizează linia READY.

În cazul sistemelor performante (memorii rapide), 8237A permite realizarea unui *transfer cu timp comprimat* (Compressed Timing), prin reducerea duratei de transfer la numai două cicluri de tact. La acest tip de transfer se folosesc numai stările S₂ și S₄ pentru generarea semnalelor de citire și scriere, starea S₃ fiind eliminată. În plus, starea S₁ este parcursă doar la modificarea octetului superior al adresei, ceea ce permite câștigarea unui număr de 255 de cicluri de tact la fiecare transfer al unui bloc de 256 de cuvinte. Evident, avantajele menționate se obțin în modurile de transfer în bloc și la cerere. În schimb, transferul de tip memorie-memorie necesită în mod normal 8 stări, deoarece pentru transferul unui octet trebuie efectuată atât o citire (cu 4 stări), cât și o scriere (tot 4 stări).

4.6.2.4. Registrele interne ale circuitului 8237A

Cunoașterea funcționalității registrelor interne permite proiectantului de sistem să programeze corespunzător dispozitivele 8237A.

Registrele de adresă și contorizare a numărului de cuvinte, de bază și curente (Base and Current Address, and Base and Current Word Count Registers) sunt prevăzute pentru fiecare canal și au dimensiunea de 16 biți (fig.4.44 și tab.4.11). Înscrierea sau citirea acestora respectă, ca și la 8257, ordinea LSB-MSB, controlată de un bistabil F/L (First/Last). În tab.4.12 sunt prezentate comenzile pentru operațiile de citire/înscriere și informațiile de pe magistrala de date. Canalele sunt specificate prin combinațiile biților A_2 și A_1 , notate în tabel cu "x x".

Tab.4.12.

		Semnale								Informația de pe
Registru	Operația	\overline{CS}	$\overline{I/OR}$	$\overline{I/OW}$	A_3	A_2	A_1	A_0	F/L	liniile $DB_7 \div DB_0$
Adresă de bază și curentă canal 0÷3	Scriere	0	1	0	0	x	x	0	0	$A_7 \div A_0$
		0	1	0	0	x	x	0	1	$A_{15} \div A_8$
Adresă curentă canal 0÷3	Citire	0	0	1	0	x	x	0	0	$A_7 \div A_0$
		0	0	1	0	x	x	0	1	$A_{15} \div A_8$
Contor de bază și curent canal 0÷ 3	Scriere	0	1	0	0	x	x	1	0	$C_7 \div C_0$
		0	1	0	0	x	x	1	1	$C_{15} \div C_8$
Contor curent canal 0÷3	Citire	0	0	1	0	x	x	1	0	$C_7 \div C_0$
		0	0	1	0	x	x	1	1	$C_{15} \div C_8$

Registrul de comandă (Command Register) este un registru de 8 biți, care controlează operarea circuitului 8237A.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
DAKCL	DREQL	EW	RP	CT	\overline{EN}	AH	MM	CW

Semnificația biților cuvântului de comandă (CW) este următoarea:

DAKCL și **DREQL** (DAK and DREQ Level) - permit alegerea nivelului activ, coborât (0) sau ridicat (1), pentru semnalele DAK_i , respectiv $DREQ_i$.

EW și **RP** au același rol ca și la controlerul 8257, permițând alegerea transferului cu *scriere extinsă* (extended write) și respectiv stabilirea modului de lucru cu *rotirea priorităților*.

CT (Compressed Timing) - permite alegerea transferului cu timp comprimat.

\overline{EN} (Enable) permite validarea/dezactivarea prin program a funcționării controlerului 8237A.

MM (Memory to Memory) - permite selectarea modului de transfer memorie-memorie, cu (**AH**=1) sau fără (**AH**=0) reținerea adresei (**AH** - Address Hold) în canalul #0.

Registrul de comandă poate fi șters (00h), fie prin resetarea hardware a circuitului, fie printr-o comandă software, denumită "Master clear".

Registrul de mod (Mode Register) este asociat fiecărui canal și are 6 biți pentru selectarea modului de operare (**M1**, **M0**), a tipului de transfer (**Rd**, **Wr**), a sensului de modificare a adreselor: cu incrementare ($\overline{I/D}$ =0) sau cu decrementare ($\overline{I/D}$ =1), precum și

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
M1	M0	$\overline{I/D}$	AI	Rd	Wr	CS1	CS0	MW
Selectie mod				Tip operație		Selectie canal		
0 0	la cerere			0 0	Verify	0 0	canal #0	
0 1	singular			0 1	Write	0 1	canal #1	
1 0	în bloc			1 0	Read	1 0	canal #2	
1 1	în cascadă			1 1	Illegal	1 1	canal #3	
				x x	dacă			
				M0 M1 = 11				

pentru a stabili dacă se utilizează autoinițializarea (AI=1). Alți doi biți, CS1 și CS0 (Channel Select), sunt folosiți pentru selectarea canalului în care se va înscrie cuvântul de mod (MW).

Registrul de cereri (Request Register). Circuitul 8237A poate răspunde la solicitări de transfer DMA inițiate atât hardware (DREQ_i), cât și software. Solicitățile lansate prin software sunt nemascabile, dar sunt supuse arbitrării blocului de codificare a priorităților, la fel ca și cele hardware. Fiecărui canal îi corespunde un bit de

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	
x	x	x	x	x	S / \bar{R}	CS1	CS0	RW
Nu contează					Set/ Reset	Selecție canal		
					Bit de cerere resetat 0	0 0 - canal #0		
					Bit de cerere setat 1	0 1 - canal #1		
						1 0 - canal #2		
						1 1 - canal #3		

cerere, într-un registru de cereri de 4 biți. Forțarea acestuia pe "1" logic, printr-un cuvânt de comandă, RW, inițiază transferul chiar dacă respectivul canal este mascat. Acest bit este resetat automat la terminarea numărării sau la activarea liniei \overline{EOP} . De asemenea, întreg registrul este șters prin RESET.

Biții CS1 și CS0 permit, și în acest caz, selectarea canalului prin care se va lansa software o cerere DMA. În fine, trebuie precizat faptul că cererile DMA inițiate software se pot face numai în modul de transfer în bloc.

Registrul de măști (Mask Register). Fiecare canal #*i* (*i*=0÷3) are asociat un bit care poate fi poziționat astfel încât canalul să ignore o cerere pe linia DREQ_i. Acești biți pot fi setați sau resetați individual sau pot fi poziționați simultan, folosind două tipuri de cuvinte de comandă:

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀		D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	
x	x	x	x	x	S / \bar{R}	CS1	CS0		x	x	x	x	M3	M2	M1	M0	
Nu contează					Set/ Reset	Sel. canal			Nu contează				#3	#2	#1	#0	
					Bit de mască resetat 0	00 - canal #0							Mi = 0 - canal #i validat				
					Bit de mască setat 1	01 - canal #1							Mi = 1 - canal #i mascat				
						10 - canal #2											
						11 - canal #3											

MW1

MW2

Dacă nu este programată autoinițializarea, la terminarea transferului printr-un canal DMA, bitul de mască aferent este setat. De asemenea, registrul este setat de semnalul RESET, care dezactivează toate cererile DMA până la transmiterea unei comenzi de ștergere a măștilor (MW1 sau MW2).

Registrul de stare (Status Register) poate fi doar citit de către microprocesor și conține informații despre starea controlerului la un moment dat (SW). Aceste informații se referă la canalele care au încheiat transferul și la cele care au cereri DMA nerezolvate încă. Astfel, biții TC3÷0 sunt setați atunci când canalele corespunzătoare au transferat numărul de cuvinte programat sau la un semnal \overline{EOP} extern. La fiecare citire a registrului de stare, precum și la activarea semnalului RESET, acești biți sunt forțați pe "0" logic. Biții DR3÷0 sunt setați ori de câte ori canalul corespunzător solicită un transfer DMA.

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	
DR3	DR2	DR1	DR0	TC3	TC2	TC1	TC0	SW
#3	#2	#1	#0	#3	#2	#1	#0	
DR _i = 0 - nu există o cerere pe canalul #i DR _i = 1 - există o cerere pe canalul #i				TC _i = 1 - canalul #i a terminat transferul				

Registrul temporar (Temporary Register) este folosit pentru citirea octetului transferat în timpul transferurilor memorie-memorie. Ultimul octet transferat poate fi citit de către microprocesor (IW), la terminarea transferului. Registrul temporar conține întotdeauna ultimul octet transferat în operația precedentă de transfer memorie-memorie, atât timp cât nu este șters prin activarea semnalului RESET.

Tab.4.13.

Registrul	Cuvântul	Operația	Semnale						
			\overline{CS}	$\overline{I/OR}$	$\overline{I/OW}$	A_3	A_2	A_1	A_0
Comandă	CW	scriere	0	1	0	1	0	0	0
Mod	MW	scriere	0	1	0	1	0	1	1
Cereri	RW	scriere	0	1	0	1	0	0	1
Măști	MW1	set / reset	0	1	0	1	0	1	0
	MW2	scriere	0	1	0	1	1	1	1
Stare	SW	citire	0	0	1	1	0	0	0
Temporar	TW	citire	0	0	1	1	1	0	1

Operațiile cu registrele de 8 biți prezentate anterior, precum și condițiile de realizare a acestora sunt prezentate, în aceeași ordine, în tab.4.13.

Tab.4.14.

Semnale						Operația efectuată
A_3	A_2	A_1	A_0	$\overline{I/OR}$	$\overline{I/OW}$	
1	0	0	0	0	1	Citire registru de stare (SW)
1	0	0	0	1	0	Scriere registru de comandă (CW)
1	0	0	1	0	1	Nepermisă
1	0	0	1	1	0	Scriere registru de cereri (RW)
1	0	1	0	0	1	Nepermisă
1	0	1	0	1	0	Setare/resetare bit în registrul de măști (MW1)
1	0	1	1	0	1	Nepermisă
1	0	1	1	1	0	Scriere registru de mod (MW)
1	1	0	0	0	1	Nepermisă
1	1	0	0	1	0	Ștergere bistabil F/L (CFL)
1	1	0	1	0	1	Citire registru temporar (TW)
1	1	0	1	1	0	Resetare software (MC)
1	1	1	0	0	1	Nepermisă
1	1	1	0	1	0	Ștergere registru măști (CM)
1	1	1	1	0	1	Nepermisă
1	1	1	1	1	0	Scriere registru de măști (MW2)

Comenzi software speciale. După cum s-a menționat anterior, controlerul 8237A acceptă un număr de comenzi software speciale, care pot fi executate în *regimul de programare*. Aceste comenzi nu depind de o anumită structură a cuvântului de pe magistrala de date, fiind decodificate de controler numai pe baza adresei de port utilizate:

- *ștergere bistabil F/L (CFL - Clear First/Last Flip-Flop)*, care se execută înaintea înscrierii sau citirii de informații noi, referitoare la adresa sau numărul de cuvinte, într-un canal al controlerului 8237A. Această comandă aduce bistabilul F/L într-o stare cunoscută, fapt ce permite microprocesorului să execute o secvență corectă de acces la registrele de 16 biți.

- *resetare software* (MC - *Master Clear*), care are același efect ca și activarea liniei RESET. La o astfel de comandă sunt resetate registrele de comandă, de stare, de cereri DMA, temporar și bistabilul F/L, iar registrul de măști este setat. Circuitul 8237A va intra în starea de inactivitate (SI).

- *ștergere registru măști* (CMR - *Clear Mask Register*), care asigură resetarea biților de mască ai celor patru canale și permite acceptarea cererilor DMA pe toate canalele.

Toate codurile comenzilor software acceptate de controlerul 8237A, împreună cu o scurtă descriere a operației efectuate de fiecare, sunt prezentate în tab.4.14.

Comparând cu tab.4.12, se constată că prin bitul A_3 se face distincția între comenzile software ($A_3=1$) și comenzile pentru registrele de 16 biți ale canalelor ($A_3=0$).

4.6.2.5. Organizarea accesului direct la memorie folosind 8237A

Organizarea accesului direct la memorie cu circuitul 8237A se realizează similar ca și cu 8257. Astfel, fiecărui controler 8237A i se atașează un registru de 8 biți, pentru reținerea octetului inferior al adresei DMA, transmis pe magistrala de date. În acest scop poate fi folosit un port 8212 sau "latch"-ul rapid 8282, ca în fig.4.46. Semnalul AEN validează ieșirea circuitului 8282 în timpul transferurilor DMA.

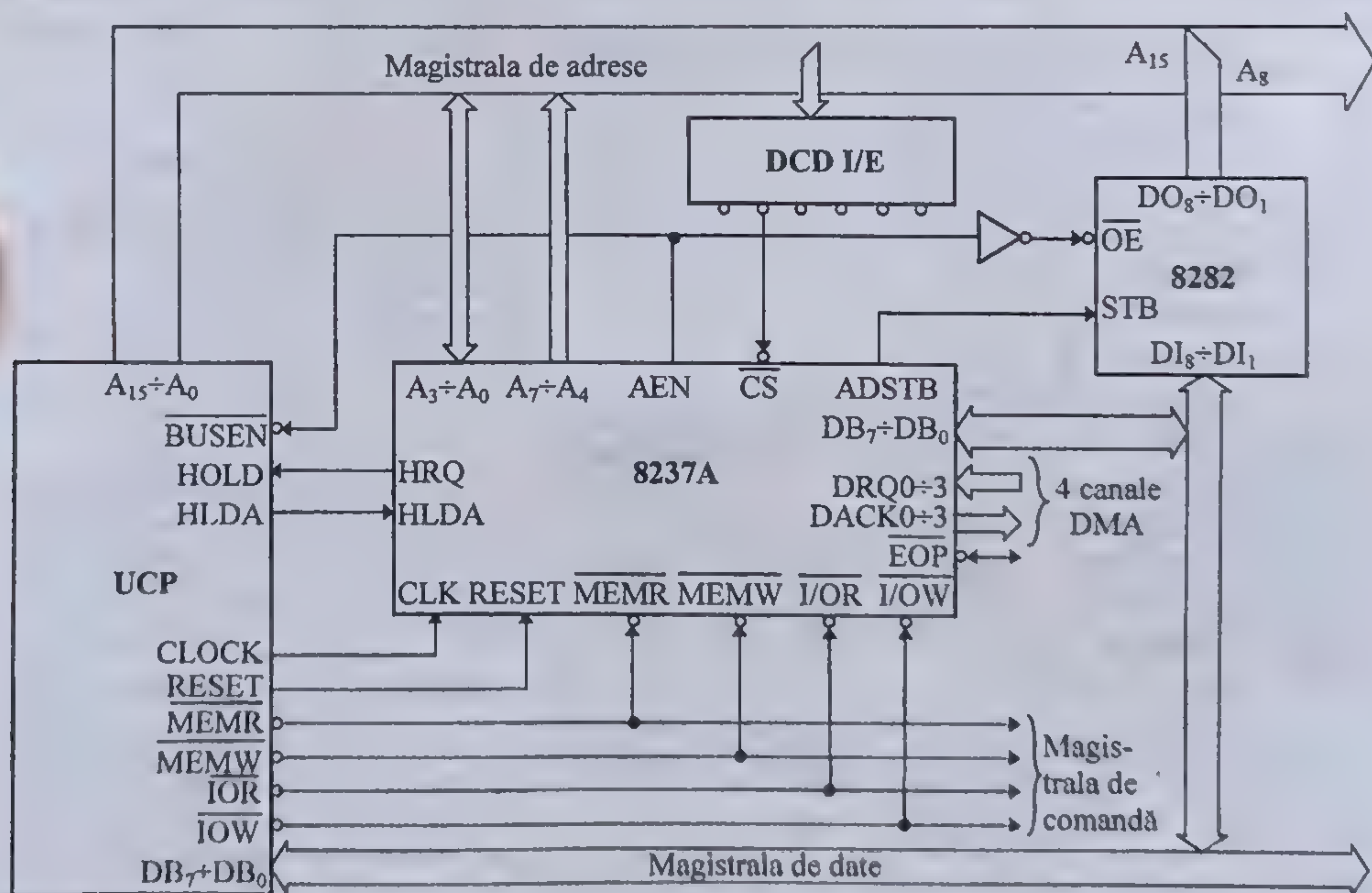


Fig.4.46. Conectarea controlerului DMA 8237A într-un sistem cu microprocesor Intel de 8 biți

După resetarea circuitului, UCP trebuie să realizeze programarea acestuia, prin înscrierea parametrilor transferului în registrele canalului corespunzător (adresa de bază, numărul de cuvinte și modul de operare) și în registrul de comenzi. Ultimul registru care se programează este registrul de măști, evitându-se astfel începerea funcționării unui canal incomplet programat. O procedură similară se respectă și la programarea unor registre ale canalelor: mai întâi se dezactivează canalul (prin setarea bitului \overline{EN} în registrul de comandă) sau se maschează canalul înaintea programării; odată programarea terminată, controlerul poate fi activat, respectiv demascat.

Este indicat ca după alimentare toate locațiile interne, în special cele ale registrelor de mod, să fie încărcate cu valori valide. Acest lucru trebuie făcut chiar dacă unele canale nu sunt utilizate [43].

Modul de utilizare a pinului $\overline{\text{EOP}}$ depinde de aplicație; dacă nu se utilizează, acesta se conectează printr-un rezistor la V_{CC} .

4.7. Interfațarea cu periferice în sisteme cu microprocesoare Intel

În majoritatea situațiilor, interfațarea cu periferice se realizează prin intermediul porturilor I/E paralele, așa cum s-a arătat în §2.1. Unul dintre cele mai versatile dispozitive de acest tip este circuitul programabil I8255/8255A, conectabil direct la magistralele sistemelor cu microprocesoare Intel.

4.7.1. Interfața paralelă 8255/8255A

Denumit *interfață programabilă cu perifericele* (PPI - *Programmable Peripheral Interface*), circuitul Intel 8255 este un port I/E, programabil, de uz general, care înlesnește conectarea perifericelor la sistemele cu microprocesoare. Este un port paralel triplu, realizat în tehnologie NMOS, compatibil TTL, încapsulat sub forma unui dispozitiv cu 40 de pini și alimentat la +5V. Circuitul este realizat în două variante: 8255 și 8255A, diferențiate prin frecvența maximă de lucru, dar identice pin cu pin și funcțional (fig.4.47) [43].

Dispozitivele 8255/8255A, sunt direct compatibile cu microprocesoarele firmei Intel de 8 și 16 biți, dar se pot utiliza și cu alte tipuri de microprocesoare (Z80).

Posedă 24 linii I/E sub forma a trei porturi de 8 biți - A, B și C - care sunt organizate în două grupuri de 12 biți: **grupul A**, format din porturile A și C superior și respectiv **grupul B**, format din B și C inferior. Utilizând 3 trei moduri de operare, familia 8255/8255A permite o interfațare facilă a UCP cu diferite tipuri de periferice.

Dialogul cu UCP se realizează prin intermediul blocului tampon al magistralei de date care, prin liniile bidirectionale $D_7 \div D_0$, primește sau transmite informații de la/spre microprocesor.

Blocul cu logica de comandă orientează liniile $D_7 \div D_0$ în corespondență cu activarea de către UCP a liniilor $\overline{\text{CS}}$ și $\overline{\text{RD}}$ (citire), respectiv $\overline{\text{WR}}$ (scriere). Acest bloc conține și *registrul cuvântului de comandă* (CWR - Control Word Register), prin intermediul căruia dispozitivul 8255 poate fi programat. Semnalul RESET aduce în zero toate registrele interne ale circuitului și configurează toate porturile ca porturi de intrare.

Pentru selectarea unuia din porturile A, B și C, respectiv a registrului de comandă (CWR), se folosesc liniile A_1 și A_0 , conectate de obicei la liniile cu același nume ale magistralei de adrese.

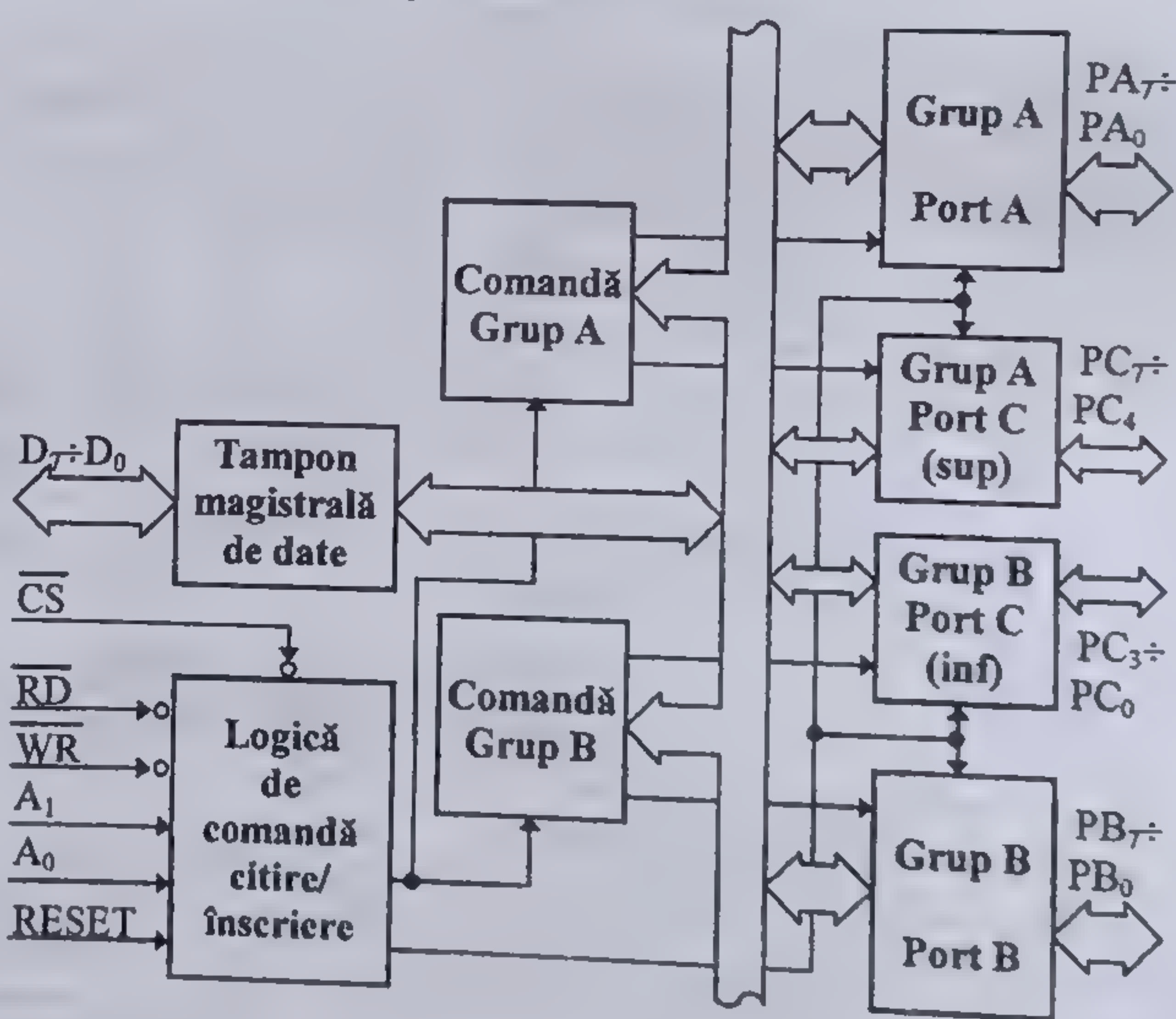


Fig.4.47. Schema bloc a circuitului 8255A

În tab.4.15 sunt prezentate operațiile care pot fi efectuate de circuitele 8255/8255A, în corespondență cu semnalele aplicate de microprocesor la intrările blocului de comandă. Se observă și în acest caz faptul că registrul cuvântului de comandă poate fi numai înscris, citirea lui nefiind permisă.

Tab.4.15

\overline{CS}	\overline{RD}	\overline{WR}	A1	A0	Tipul operației efectuate
0	0	1	0	0	Citire port A
0	0	1	0	1	Citire port B
0	0	1	1	0	Citire port C
0	0	1	1	1	Nici o operație (D7÷D0 în HZ)
0	1	0	0	0	Scriere în portul A
0	1	0	0	1	Scriere în portul B
0	1	0	1	0	Scriere în portul C
0	1	0	1	1	Scriere în registrul de comandă
1	x	x	x	x	Nici o operație (D7÷D0 în HZ)
0	1	1	x	x	Nici o operație (D7÷D0 în HZ)

În afara semnalelor de control generate de logica de citire/înscris, cele două blocuri de comandă a grupurilor generează și ele comenzi pentru porturile I/E.

Fiecare din cele trei porturi I/E poate fi utilizat independent și posedă facilități specifice [14, 43]:

- portul A are un tampon de ieșire cu memorare și un tampon cu memorare pe intrare;
- portul B are un tampon intrare/ieșire cu memorare și un tampon de intrare;
- portul C are un tampon de intrare fără memorare și un tampon de ieșire cu memorare.

În plus, portul C poate fi controlat atât la nivel de semiport, cât și la nivel de bit, fiecare linie putând fi activată sau dezactivată individual (Bit Set/Reset Operation). În acest fel, liniile portului C pot fi folosite ca biți de comandă/stare pentru transferul datelor prin porturile A și B.

4.7.1.1. Moduri de operare

Circuitul 8255A poate lucra în trei moduri, selectabile prin program.

Modul 0 - intrare/ieșire nesincronizată - permite realizarea transferului paralel de date fără confirmare, prin cele trei porturi de 8 biți: A, B și C. Porturile pot fi programate în 16 combinații, fie ca porturi de intrare, fie ca porturi de ieșire (portul C programat la nivel de semiport). În fig.4.48 este reprezentată schematic operarea în modul 0 și conectarea circuitului la magistralele sistemului. Datele sunt înscrise sau citite în/din porturi la inițiativa UCP. Caracteristicile principale ale acestui mod sunt: ieșirile sunt cu memorare, iar intrările sunt fără memorare (μP preia datele găsite la pini în momentul citirii).

Modul 1 - intrare/ieșire sincronizată - asigură transferul I/E cu confirmare (strobed I/O mode) prin porturile A și B; acestea utilizează, atât la intrare cât și la ieșire, registre cu memorare. Liniile portului C sunt rezervate pentru semnale de control al transferului I/E și sunt repartizate celor două porturi, corespunzător funcționalității acestora: porturi de intrare (I) sau de ieșire (E). Ambele porturi pot genera întreruperi pe liniile de comandă, $INTR_A$ și respectiv $INTR_B$.

În cazul în care A și/sau B sunt porturi de intrare, liniile de confirmare sunt denumite \overline{STB} (STroBe) și IBF (Input Buffer Full) (fig.4.48). După ce depune data pe liniile de intrare (momentul 1 în fig.4.49), perifericul comandă semnalul \overline{STB} pe nivel "0" logic. În acest moment, data de la periferic este încărcată în port și memorată (2). Confirmarea încărcării se face prin linia IBF , care trece în "1" logic (3). Informația despre încărcarea datei, perifericul dezactivează linia \overline{STB} (4) și poate înceta să mai mențină data validă pe liniile de intrare (5). În momentul în care este îndeplinită condiția $\overline{STB} \cdot \overline{IBF} \cdot \overline{RD} = 1$, circuitul 8255 lansează o cerere de întrerupere (6) către UCP, prin liniile $PC_3 = INTR_A$ (pentru portul A) și respectiv $PC_0 = INTR_B$ (pentru portul B).

Înterupterile sunt mascabile prin două bistabile, $INTE_A$ și respectiv $INTE_B$, comandate în regim "bit set/reset", pe liniile PC_4 ($INTE_A$) și respectiv PC_2 ($INTE_B$).

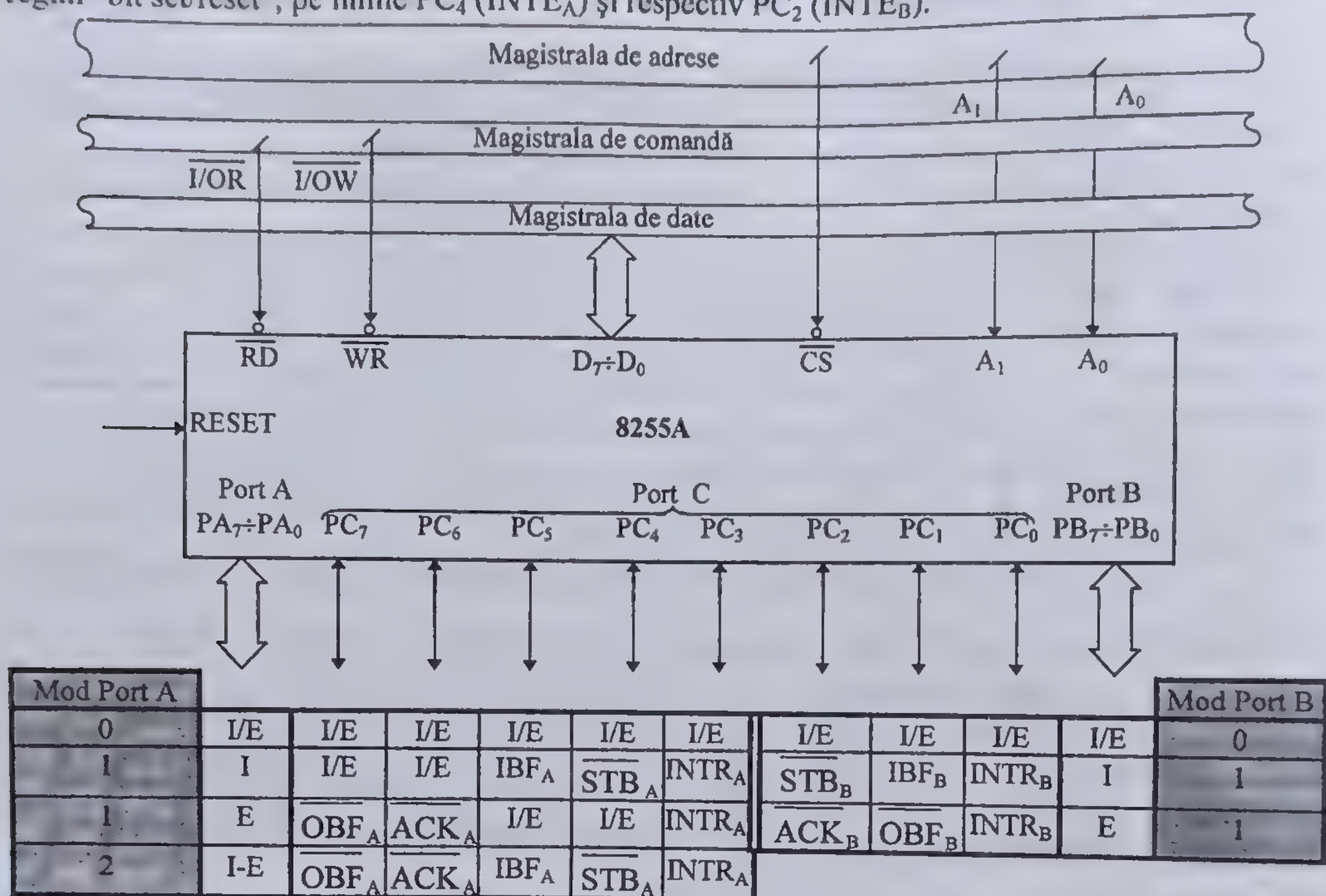


Fig.4.48. Conectarea PPI 8255A în sistem și semnificația pinilor în cele trei moduri de operare

În momentul în care UCP află, prin întrerupere sau prin testarea software a bitului INTR corespunzător, că data de la periferic este disponibilă în portul de intrare, declanșează o procedură de citire a portului. Pe frontul căzător al semnalului \overline{RD} (7) se dezactivează semnalul INTR (8), este citită data din port, iar cu frontul crescător (9) se dezactivează și semnalul IBF (10). Astfel, perifericul este informat că portul este liber și că poate să înscrie o nouă dată, respectând aceeași procedură.

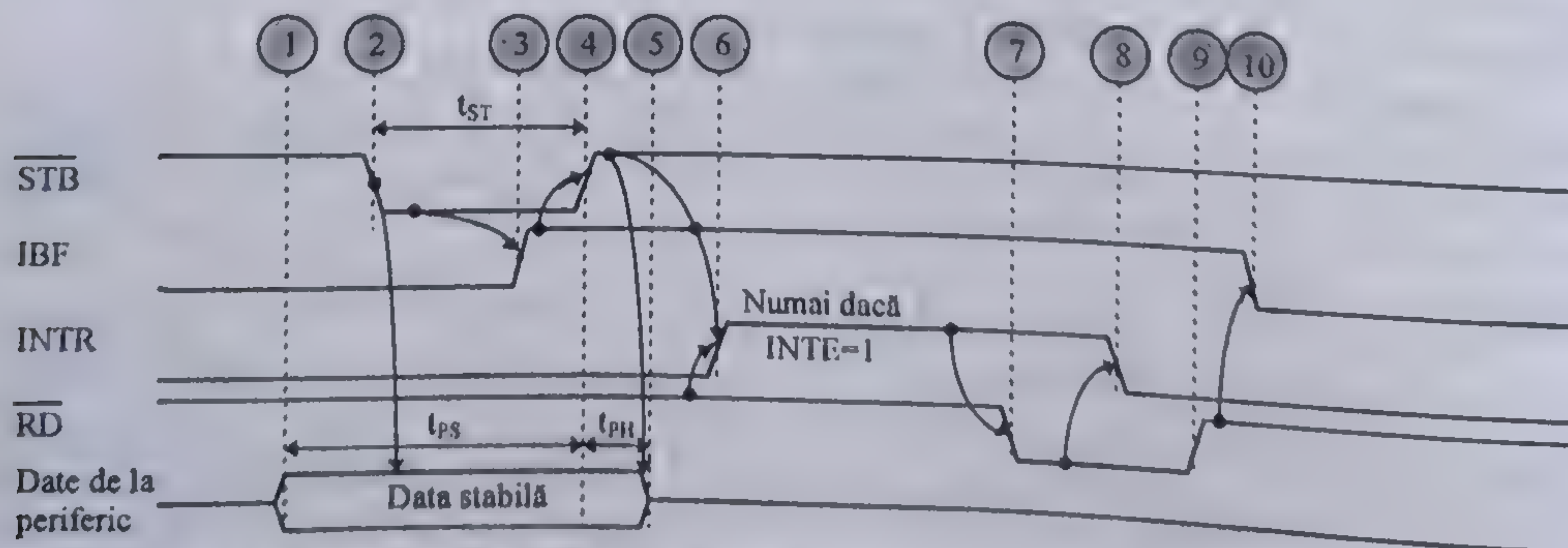


Fig.4.49. Evoluția semnalelor de control în modul 1; A, B - porturi de intrare

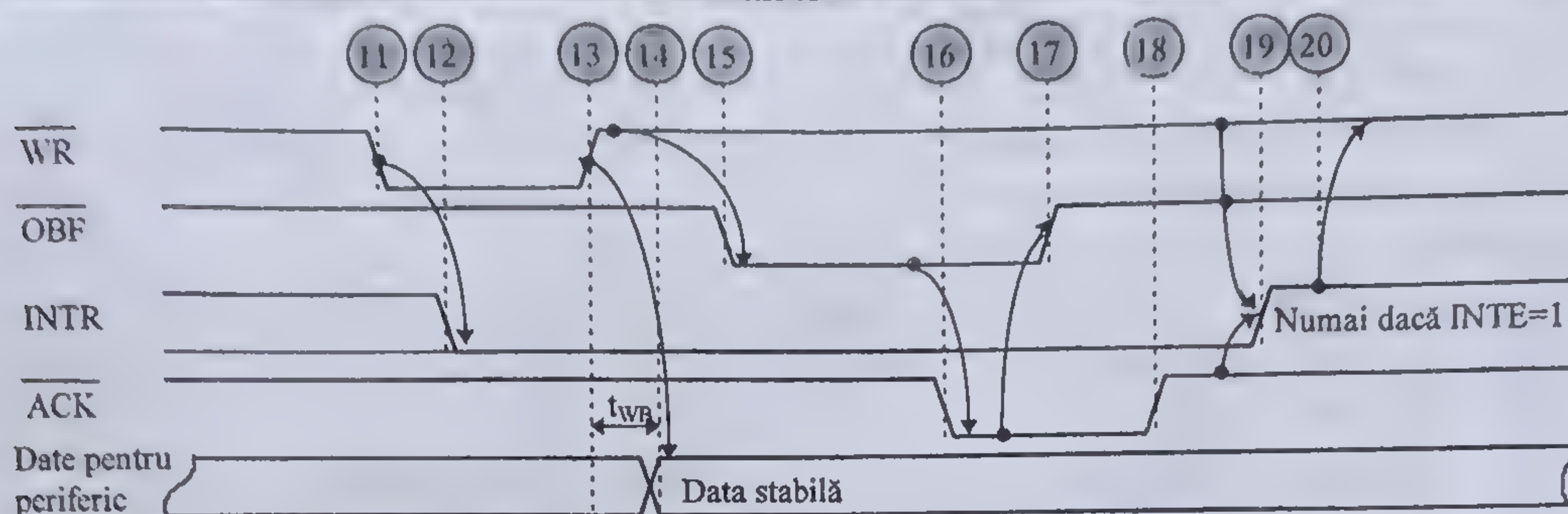


Fig.4.50. Evoluția semnalelor de control în modul 1; A, B - porturi de ieșire

Pentru ca informația să fie corect memorată în portul de intrare, lățimea impulsului de "strobe" (t_{ST}) nu poate să scadă sub o valoare minimă (500ns), iar datele de la periferic trebuie să fie menținute stabile după strobare, un timp $t_{PH} \geq 180ns$.

În cazul în care A și/sau B sunt *porturi de ieșire*, liniile de control sunt denumite \overline{OBF} (Output Buffer Full) și \overline{ACK} (ACKnowledge) (fig.4.48) și au rolul de a indica prezența datei valide în portul de ieșire, respectiv de confirmare, de către periferic, a preluării datei din port. Microprocesorul înscrie data în portul de ieșire (momentul 11 în fig.4.50), care se regăsește pe liniile acestuia după o întârziere $t_{WB} = \max. 350ns$ (14) de la frontul crescător al semnalului \overline{WR} . În același timp (12), se dezactivează cererea de întrerupere pe linia INTR (dacă există), iar după dezactivarea semnalului \overline{WR} (13) se activează linia \overline{OBF} (15), care anunță perifericul că datele sunt valide pe liniile de ieșire ale portului. La rândul său, perifericul confirmă preluarea datei (16), prin activarea semnalului \overline{ACK} (min. 300ns); aceasta determină portul să-și dezactiveze linia \overline{OBF} (17). La îndeplinirea condiției $\overline{WR} \cdot \overline{OBF} \cdot \overline{ACK} = 1$ (18), apare o nouă cerere de întrerupere pe linia INTR (19). Aceasta anunță UCP că portul de ieșire este pregătit să accepte noi date pentru periferic (20).

Și în acest caz întreruperile sunt mascabile, prin bistabilele $INTE_A$ și $INTE_B$, de această dată prin comenzi "bit set/reset" pe liniile PC_6 ($INTE_A$) și respectiv PC_2 ($INTE_B$).

Modul 2 - intrare-ieșire sincronizată - se caracterizează prin realizarea unui transfer cu confirmare, bidirecțional, posibil numai prin portul A (strobed bidirectional bus I/O). Pentru funcționarea în acest mod, portul A utilizează un număr de 5 linii ale portului C (fig.4.48). Acestea asigură semnalele de comandă pentru intrarea (\overline{STB}_A și IBF_A), respectiv pentru ieșirea datelor (\overline{OBF}_A și \overline{ACK}_A), ambele direcții fiind cu memorare.

Transferul bidirecțional poate fi realizat prin întreruperi, când prin linia $INTR_A$ se pot lansa cereri atât pentru intrare, cât și pentru ieșire ($INTR_A = \overline{STB}_A \cdot IBF_A \cdot \overline{RD} + \overline{WR} \cdot \overline{OBF}_A \cdot \overline{ACK}_A$). Întreruperile sunt și în acest caz mascabile, prin două bistabile de întreruperi denumite aici $INTE_1$ și $INTE_2$, controlate pe liniile PC_6 și respectiv PC_4 .

Liniile rămase, PC_2+PC_0 , pot fi folosite fie ca linii de intrare/ieșire în modul 0, fie ca linii de control pentru portul B, atunci când acesta lucrează în modul 1.

Evoluția semnalelor în modul 2 este prezentată în fig.4.51; momentele semnificative sunt marcate cu numerele corespunzătoare modului 1. Operațiile de intrare și de ieșire se pot efectua în orice secvență în care semnalul \overline{ACK} este activat după înscrierea unui octet de către microprocesor ($\overline{WR}=0$), iar citirea unui octet de către microprocesor ($\overline{RD}=0$) urmează unei

activări a semnalului \overline{STB}_A . Microprocesorul poate oricând să înscrie date în portul A, chiar și în același timp cu perifericul. Data înscrisă de către microprocesor (11) va apare pe liniile $PA_7 \div PA_0$ (14) abia după ce perifericul activează linia \overline{ACK}_A (16), într-un interval t_{AD} . Datele rămân numai un timp t_{KD} după dezactivarea liniei \overline{ACK}_A . Linia \overline{INTR}_A este activată de operația de intrare de la periferic (6), când $\overline{STB}_A \cdot \overline{IBF}_A \cdot \overline{RD} = 1$. După terminarea operației de ieșire se mai adaugă și condiția $\overline{WR} \cdot \overline{OBF}_A \cdot \overline{ACK}_A = 1$ (19). După preluarea datei de către microprocesor, $\overline{STB}_A \cdot \overline{IBF}_A \cdot \overline{RD} = 0$, rămâne valabilă condiția $\overline{WR} \cdot \overline{OBF}_A \cdot \overline{ACK}_A = 1$ (8), deci \overline{INTR}_A rămâne activată în continuare (20).

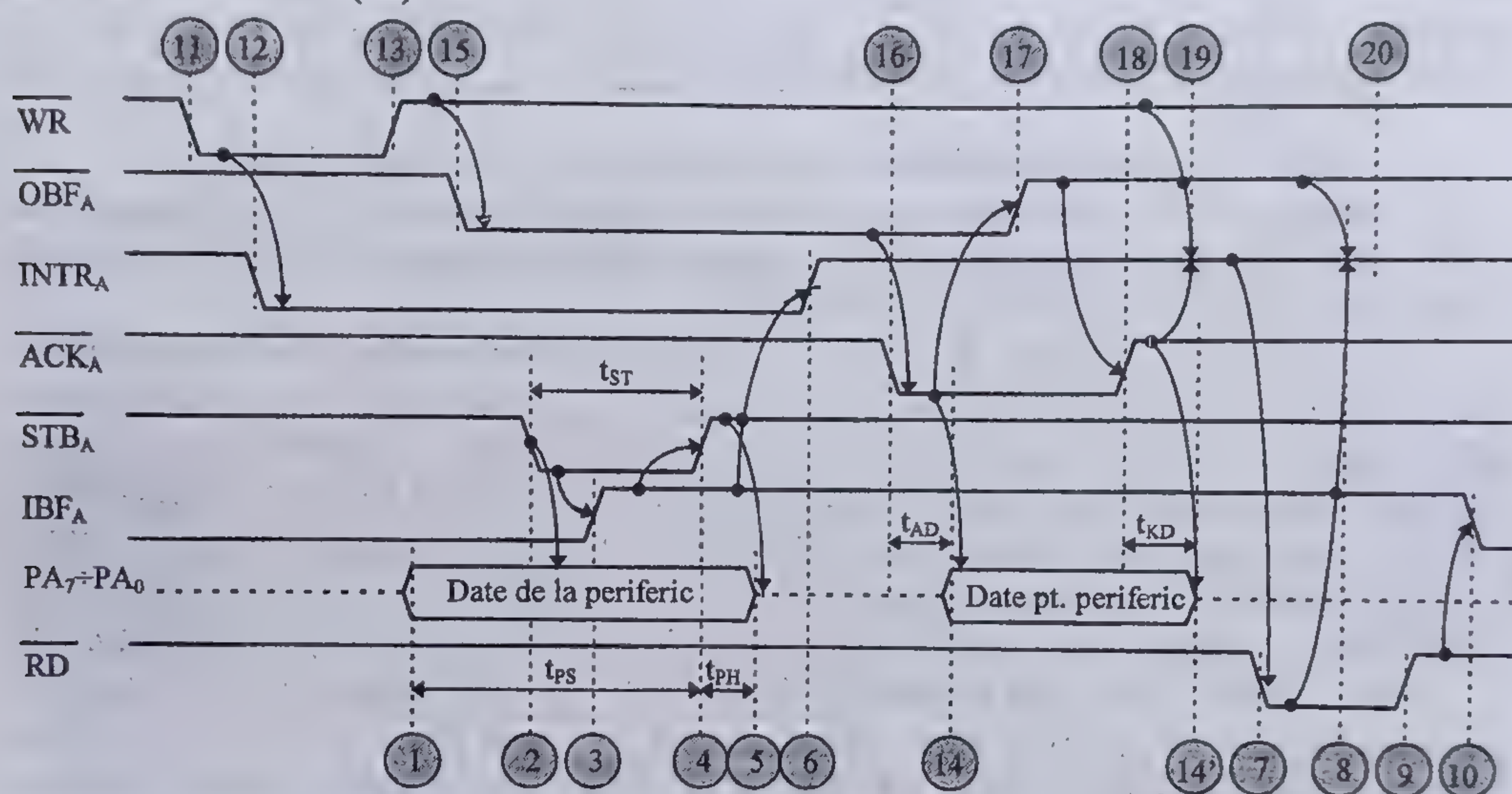


Fig.4.51: Evoluția semnalelor de control în modul 2; A - port intrare-ieșire

Controlul fluxului de date pe liniile $PA_7 \div PA_0$ cade în sarcina perifericului. Pentru a evita apariția unor conflicte pe aceste linii, perifericul de I/E trebuie să activeze, pe rând, semnalele \overline{STB}_A (după ce în prealabil a depus data pe liniile $PA_7 \div PA_0$) și \overline{ACK}_A (când data din tamponul de ieșire al portului apare pe liniile $PA_7 \div PA_0$). În acest scop, perifericul utilizează semnalele de control generate de port, \overline{IBF}_A și \overline{OBF}_A .

4.7.1.2. Programarea dispozitivului 8255A

Programarea circuitului vizează stabilirea modului de lucru (0, 1 sau 2) și a tipului (I - intrare, E - ieșire) pentru fiecare port. În acest scop, trebuie înscris în registrul de comandă al circuitului (v.tab.4.15) un cuvânt de comandă, CW (Control Word), având următoarea structură:

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	
1	MA ₁	MA ₀	PA	PCS	MB	PB	PCI	CW
Grup A					Grup B			
Mod			Tip		Mod	Tip		
Port A			Port A	PC ₇ ÷PC ₄	Port B	Port B	PC ₃ ÷PC ₀	
0	0	-	modul 0	1 - I	1 - I	0 - mod 0	1 - I	1 - I
0	1	-	modul 1	0 - E	0 - E	1 - mod 1	0 - E	0 - E
1	x	-	modul 2					

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

Selectarea modului de lucru se face separat pentru cele două porturi principale, A și B, prin MA_1 , MA_0 , respectiv MB . În modul 0, porturile pot fi programate independent, ca porturi de intrare sau de ieșire. Pentru modurile 1 și 2 (cel din urmă numai la portul A), o parte din liniile portului C sunt alocate automat porturilor A și B, ca semnale de control (hardware handshaking), așa cum s-a arătat mai înainte (v.fig.4.48). Liniile de la portul C rămase nefolosite pot fi, de asemenea, programate ca linii de intrare sau de ieșire, prin PCS și PCI.

Cuvântul de comandă pentru stabilirea modului de lucru este recunoscut ca atare de 8255A numai dacă bitul cel mai semnificativ (D_7) este "1" logic. În caz contrar ($D_7=0$), cuvântul este interpretat ca o comandă de tip "single bit set/reset" (SBSR) pentru biții portului C, având următoarea configurație:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
0	x	x	x	B_2	B_1	B_0	S / R	SBSR
Nu contează				Selecție bit port C:			Operația	
				PC_0	0	0	0	1 - Set 0 - Reset
				PC_1	0	0	1	
				PC_2	0	1	0	
				PC_3	0	1	1	
				PC_4	1	0	0	
				PC_5	1	0	1	
				PC_6	1	1	0	
				PC_7	1	1	1	

Această facilitate este avantajoasă atunci când sunt necesare comenzi la nivel de bit, precum și pentru activarea/dezactivarea întreruperilor, în modurile 1 și 2, prin setarea/resetarea biților INTE corespunzători (vezi SW). Ambele tipuri de cuvinte se transmit registrului de comandă, în conformitate cu tab.4.15 ($A_1=A_0=1$), în cadrul unor operații de înscriere ($\overline{RD}=1$, $\overline{WR}=0$).

În plus, atunci când 8255A este programat în modurile 1 sau 2, prin citirea portului C se obține pe magistrala de date o *informație de stare*, SW (Status Word), atât despre semnalele de confirmare a transferului (v.fig.4.48), cât și despre bistabilele de întrerupere.

Mod Port A	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Mod Port B
0 - intrare/ieșire	PC_7	PC_6	PC_5	PC_4	PC_3	PC_2	PC_1	PC_0	0 - intrare/ieșire
1 - intrare	PC_7	PC_6	IBF_A	$INTE_A$	$INTR_A$	$INTE_B$	IBF_B	$INTR_B$	1 - intrare
1 - ieșire	\overline{OBF}_A	$INTE_A$	PC_5	PC_4	$INTR_A$	$INTE_B$	\overline{OBF}_B	$INTR_B$	1 - ieșire
2 - intrare-ieșire	\overline{OBF}_A	$INTE_1$	IBF_A	$INTE_2$	$INTR_A$	SW			

Prin citirea cuvântului de stare, microprocesorul poate verifica "starea" porturilor A și/sau B. Astfel se poate afla, de exemplu, dacă există un octet nepreluat de la un periferic de intrare ($IBF=1$, $INTR=1$) sau dacă perifericul a preluat data din portul de ieșire ($\overline{OBF}=1$, $INTR=1$), pentru a se putea înscrie un nou octet.

4.7.2. Utilizarea circuitului 8255/8255A

Din cele prezentate până aici, rezultă că dispozitivul 8255A asigură toate condițiile necesare pentru interfașarea paralelă cu periferice, cu sau fără semnale de confirmare. De

Selectarea modului de lucru se face separat pentru cele două porturi principale, A și B, prin MA_1 , MA_0 , respectiv MB . În modul 0, porturile pot fi programate independent, ca porturi de intrare sau de ieșire. Pentru modurile 1 și 2 (cel din urmă numai la portul A), o parte din liniile portului C sunt alocate automat porturilor A și B, ca semnale de control (hardware handshaking), așa cum s-a arătat mai înainte (v.fig.4.48). Liniile de la portul C rămase nefolosite pot fi, de asemenea, programate ca linii de intrare sau de ieșire, prin PCS și PCI.

Cuvântul de comandă pentru stabilirea modului de lucru este recunoscut ca atare de 8255A numai dacă bitul cel mai semnificativ (D_7) este "1" logic. În caz contrar ($D_7=0$), cuvântul este interpretat ca o comandă de tip "single bit set/reset" (SBSR) pentru biții portului C, având următoarea configurație:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
0	x	x	x	B_2	B_1	B_0	S / R	SBSR
Nu contează				Selectie bit port C:			Operația	
				PC_0	0	0	0	1 - Set 0 - Reset
				PC_1	0	0	1	
				PC_2	0	1	0	
				PC_3	0	1	1	
				PC_4	1	0	0	
				PC_5	1	0	1	
				PC_6	1	1	0	
				PC_7	1	1	1	

Această facilitate este avantajoasă atunci când sunt necesare comenzi la nivel de bit, precum și pentru activarea/dezactivarea întreruperilor, în modurile 1 și 2, prin setarea/resetarea biților INTE corespunzători (vezi SW). Ambele tipuri de cuvinte se transmit registrului de comandă, în conformitate cu tab.4.15 ($A_1=A_0=1$), în cadrul unor operații de înscriere ($\overline{RD}=1$, $\overline{WR}=0$).

În plus, atunci când 8255A este programat în modurile 1 sau 2, prin citirea portului C se obține pe magistrala de date o *informație de stare*, SW (Status Word), atât despre semnalele de confirmare a transferului (v.fig.4.48), cât și despre bistabilele de întrerupere.

Mod Port A	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Mod Port B
0 - intrare/ieșire	PC_7	PC_6	PC_5	PC_4	PC_3	PC_2	PC_1	PC_0	0 - intrare/ieșire
1 - intrare	PC_7	PC_6	IBF_A	$INTE_A$	$INTR_A$	$INTE_B$	IBF_B	$INTR_B$	1 - intrare
1 - ieșire	\overline{OBF}_A	$INTE_A$	PC_5	PC_4	$INTR_A$	$INTE_B$	\overline{OBF}_B	$INTR_B$	1 - ieșire
2 - intrare-ieșire	\overline{OBF}_A	$INTE_1$	IBF_A	$INTE_2$	$INTR_A$	SW			

Prin citirea cuvântului de stare, microprocesorul poate verifica "starea" porturilor A și/sau B. Astfel se poate afla, de exemplu, dacă există un octet nepreluat de la un periferic de intrare ($IBF=1$, $INTR=1$) sau dacă perifericul a preluat data din portul de ieșire ($\overline{OBF}=1$, $INTR=1$), pentru a se putea înscrie un nou octet.

4.7.2. Utilizarea circuitului 8255/8255A

Din cele prezentate până aici, rezultă că dispozitivul 8255A asigură toate condițiile necesare pentru interfațarea paralelă cu periferice, cu sau fără semnale de confirmare. De

asemenea, posedă o flexibilitate remarcabilă, care permite proiectantului de sistem adoptarea celor mai avantajoase moduri de operare, în funcție de cerințele aplicației.

În fig.4.52 și fig.4.53 sunt prezentate două exemple de aplicații folosind acest circuit: interfațarea cu tastaturi și afișaje cu electronică proprie, respectiv cu convertoare D/A și A/D. Din examinarea caracteristicilor de interfață ale perifericelor, atât în ceea ce privește transferul datelor, cât și secvențierea temporală a operațiilor, se pot stabili, pentru fiecare caz în parte, modurile de lucru și cuvintele de comandă necesare pentru programarea circuitului 8255A.

Pentru primul exemplu, tastatura furnizează codul tastei acționate dintr-o matrice cu 64 de taste (2^6), pe liniile $R_5 \div R_0$, în paralel cu starea a două taste funcționale: SHIFT și CTRL (Control). Codul de 6 biți se transmite în exterior odată cu activarea semnalului \overline{STB} (conectat la \overline{STB}_A). Preluarea informației din portul de intrare (de la tastatură) trebuie semnalată pe linia ACK (conectat la \overline{IBF}_A).

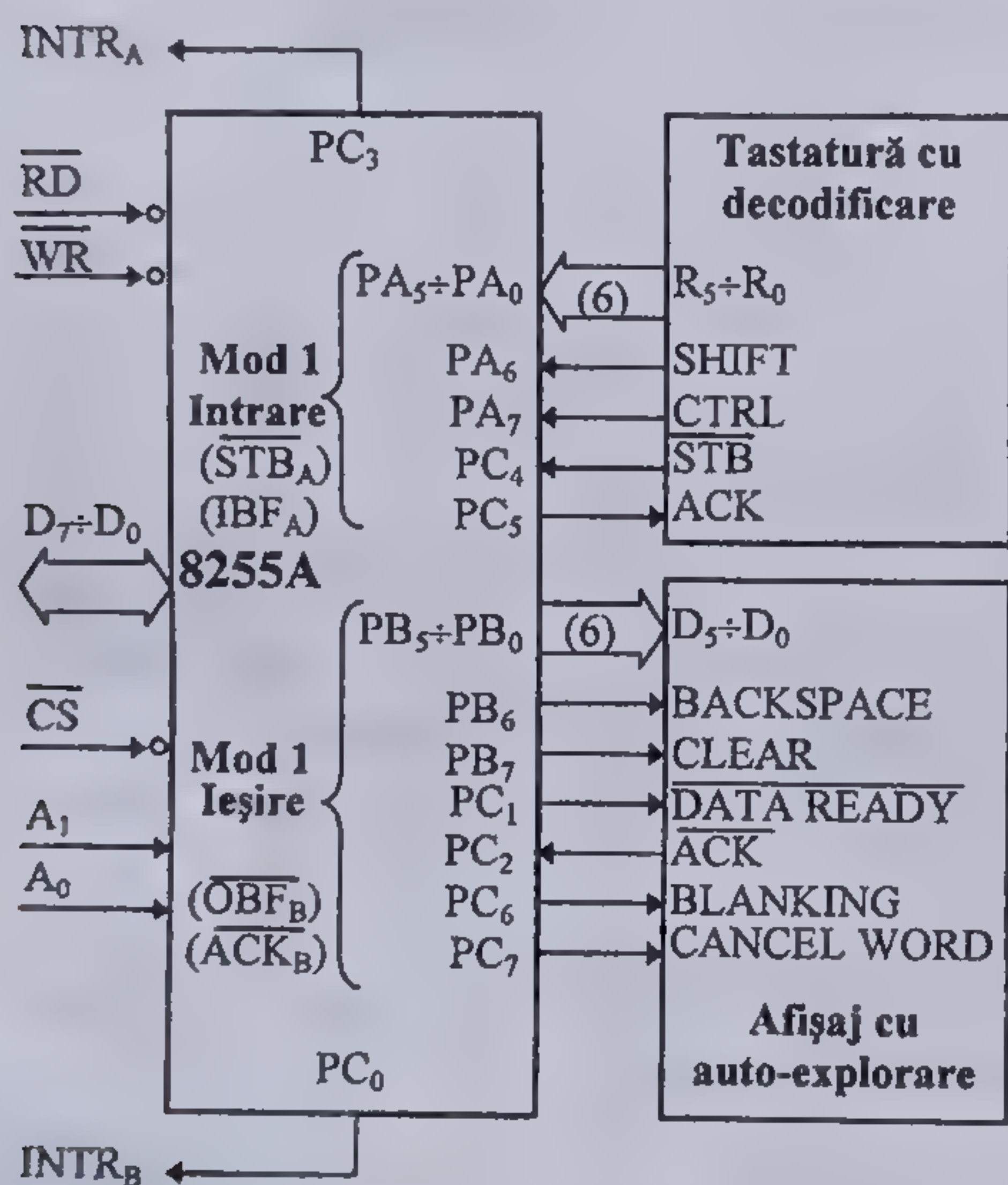


Fig.4.52. Interfațarea cu tastatură și afișaj cu logică de control proprie

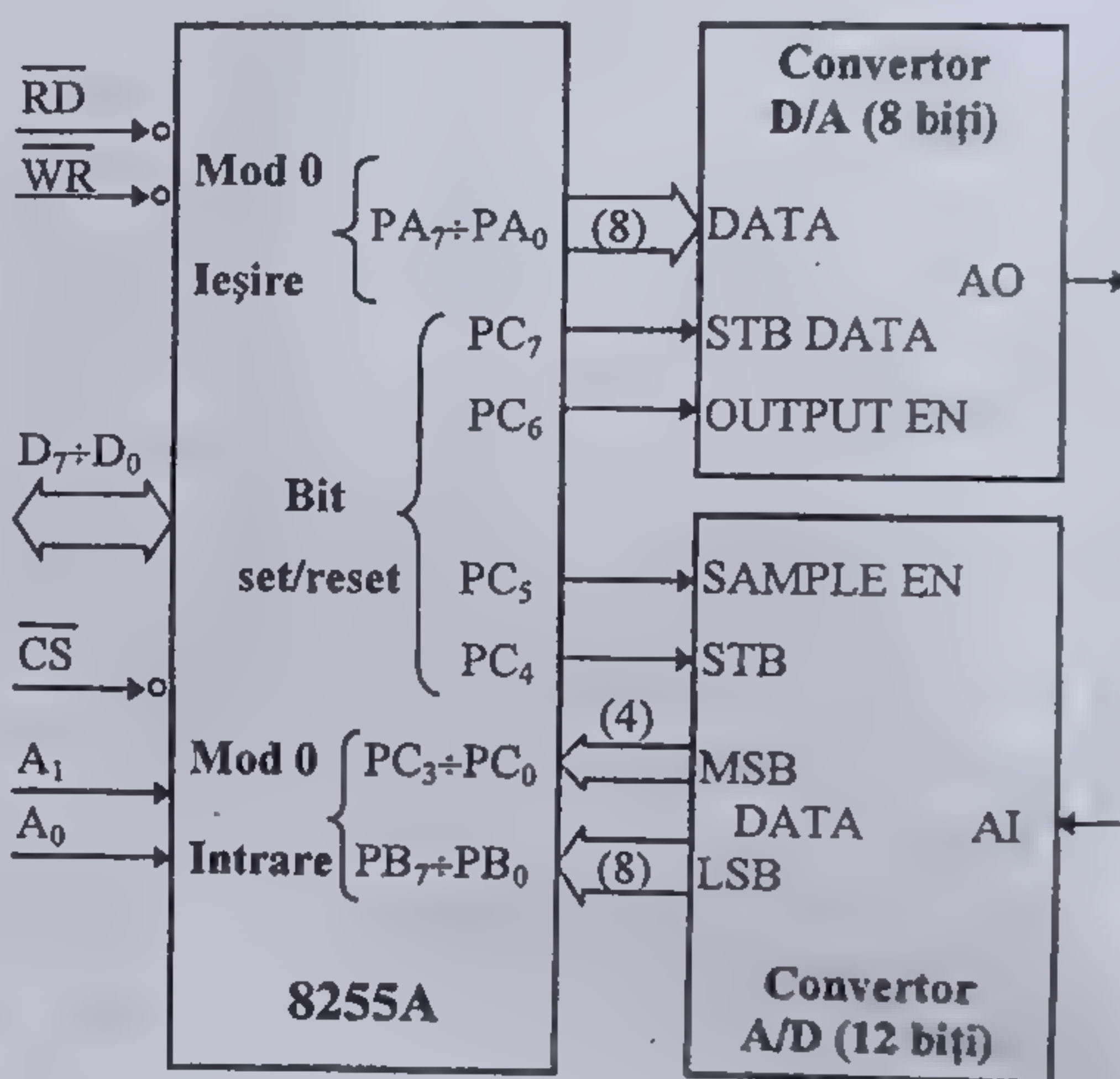


Fig.4.53. Interfațarea cu convertoare D/A și A/D

Pentru o tratare în timp real a unui astfel de periferic de către UCP este bine să se lucreze cu întreruperi. Ca urmare, cel mai avantajos este modul 1, cu generarea cererilor de întrerupere la fiecare nouă informație furnizată de tastatură; pentru aceasta se folosește portul A în regim de intrare. Similar se poate alocă portul B, tot în modul 1 - dar în regim de ieșire, cu folosirea întreruperilor pentru comanda unui afișaj cu auto-explorare (self-scan display).

Liniile PC_6 și PC_7 sunt folosite pentru comenzi asincrone de spațiere (BLANKING) sau de ștergere (CANCEL WORD), în regim de comandă pe bit. Rezultă, în acest caz, următorul cuvânt de comandă (CW1):

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	1	1	0	1	0	x
Port A - Modul 1		Port A - Intrare		PC_6, PC_7 - Ieșire		Port B - Modul 1	PC_3+PC_0 - indiferent

CW1

Liniile PC_5+PC_0 sunt alocate automat porturilor A și B, în conformitate cu fig.4.48, deci nu vor fi influențate de biții D_3 și D_0 din cuvântul de comandă CW1; numai cei doi biți care au mai rămas, PC_6 și PC_7 , vor fi programați ca ieșire, prin $D_3=0$.

De asemenea, pentru activarea liniilor $INTR_A$ și $INTR_B$, trebuie setate bistabilele $INTE_A$ și $INTE_B$, prin cuvinte de tip SBSR, pentru biții PC_4 și respectiv PC_2 . Considerând că adresa circuitului 8255A este 50h, rezultă următoarea secvență de instrucțiuni pentru programarea acestuia:

```
; PROGRAMARE 8255A PENTRU INTERFAȚAREA CU TASTATURA ȘI CU AFISAJ
; CU ELECTRONICĂ ÎNGLOBATĂ
PPI    EQU    50h           ; Adresa de bază 8255A.
CW1    EQU    10110100b    ; Cuvântul de comandă pentru configurare 8255A.
SINTEA EQU    00001001b    ; Comandă setare bit  $PC_4=INTE_A$ .
SINTEB EQU    00000101b    ; Comandă setare bit  $PC_2=INTE_B$ .

MVI    A,CW1              ; Încarcă în acumulator cuvântul de comandă.
OUT     PPI+3              ; Înscrie CW1 în registrul de comandă (adresa 53h)
MVI    A,SINTEA           ; Încarcă în acumulator comanda de setare bistabil  $INTE_A$ .
OUT     PPI+3              ; Validare generare întreruperi pentru portul A.
MVI    A,SINTEB           ; Validare întreruperi de la portul B.
OUT     PPI+3              ; Validează întreruperile la nivelul microprocesorului.
EI

; COMANDĂ SPATIU INTRE CARACTERE
MVI    A,00001101b        ; Setează linia  $PC_6$ .
OUT     PPI+3              ; Este tot un cuvânt de comandă, deci se înscrie tot la 53h.
MVI    A,00001100b        ; Resetează linia  $PC_6$ .
OUT     PPI+3

; COMANDĂ STERGerea CUVÂNTULUI AFISAT
MVI    A,0Fh              ; Setează linia  $PC_7$ .
OUT     PPI+3
MVI    A,0Eh              ; Resetează linia  $PC_7$ .
OUT     PPI+3
```

Pentru aplicația din fig.4.53, conversia A/D se face pe 12 biți. Aceasta impune folosirea completă a unui grup de la 8255A, în cazul de față grupul B, în regim de intrare. Lansarea conversiei (SAMPLE ENable) se face de către UCP, prin intermediul circuitului PPI 8255A (linia PC_5), la intervale de timp egale cu perioada de eșantionare. Citirea rezultatului conversiei se face printr-o comandă STB (dată prin PC_4), ținând cont de durata conversiei.

Conversia D/A are o rezoluție de 8 biți. UCP încarcă datele în convertor prin portul A, la activarea liniei STB DATA (prin PC_7), după care comandă validarea ieșirii, OUTPUT Enable (prin PC_6). Rezultă, pentru toate porturile, o funcționare în modul 0 și următorul cuvânt de comandă pentru 8255A (CW2):

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	0	0	0	0	0	1	1	CW2
Port A - Modul 0		Port A - Ieșire		PC_7+PC_4 - Ieșire		Port B - Modul 0	Port B - Intrare	PC_3+PC_0 - Intrare

Datorită limitării impuse magistralei de date de microprocesoarele de 8 biți, pentru extragerea rezultatului conversiei A/D UCP trebuie să efectueze două citiri succesive din 8255A: din portul B (cei mai puțin semnificativi 8 biți) și din portul C (cei mai semnificativi 4 biți).

4.8. Comunicația serială în sisteme cu microprocesoare Intel

Datorită simplității sale, comunicația serială este larg folosită pentru schimbul de informații. Ea se utilizează acolo unde viteza de transfer nu reprezintă o condiție critică, sau în cazul distanțelor mari între transmitător și receptor (sute de metri), unde transferul paralel nu este aplicabil. În sistemele cu microprocesoare Intel se utilizează mult circuitul specializat **I8251A**, care asigură transmisia/recepția serie de tip asincron sau sincron - USART (Universal Synchronous/Asynchronous Receiver/Transmitter).

4.8.1. Interfața serială 8251/8251A

Considerat o interfață de comunicație programabilă (PCI - *Programmable Communication Interface*), 8251A este versiunea îmbunătățită a dispozitivului 8251, cu care este direct compatibil. În modul asincron, acesta permite o viteză de comunicație maximă de 19,2KBaud, față de numai 9,6KBaud la 8251, iar în modul sincron max. 64KBaud, față de 56Kbaud la 8251. Realizat sub forma unui circuit integrat cu 28 pini, 8251A se alimentează la +5V și are schema bloc din fig.4.54 [43].

Dispozitivul este proiectat să funcționeze atât cu microprocesoarele de 8 cât și cu cele de 16 biți ale firmei

Intel, pentru transferul serial de cuvinte de 5÷8 biți. Comunicația cu microprocesorul se face în format paralel, pe 8 biți, prin intermediul blocului tampon al magistralei de date.

Logica de comandă și citire/înscrisere asigură controlul și sincronizarea transmiției și recepției datelor cu funcționarea microprocesorului. Sincronizarea se realizează prin semnalul CLK, care se conectează de obicei la semnalul de tact al microprocesorului. Pentru o corectă funcționare a transferului serial este necesar ca frecvența semnalului CLK, cuprinsă între 750 KHz și 3 MHz, să fie de cel puțin 30 de ori mai mare decât debitul binar la recepție/transmisie.

Semnalul RESET aduce circuitul 8251A într-o stare de inactivitate (idle), în care se menține până la programare. Durata minimă a acestui semnal trebuie să fie de 6 cicluri de tact ($t_{cy} = 0,32 \div 1,35 \mu s$). Linia C/\bar{D} (Control/Data), împreună cu \bar{CS} , \bar{RD} și \bar{WR} , permite selectarea tipului de informație vehiculată pe magistrala $D_7 \div D_0$. În tab.4.16 se prezintă efectul acestor semnale de comandă.

Obișnuit, linia C/\bar{D} se conectează la linia A_0 a magistralei de adrese. Transmisia și recepția serială a datelor se face prin două canale, prevăzute cu dublu tampon, complet independen-

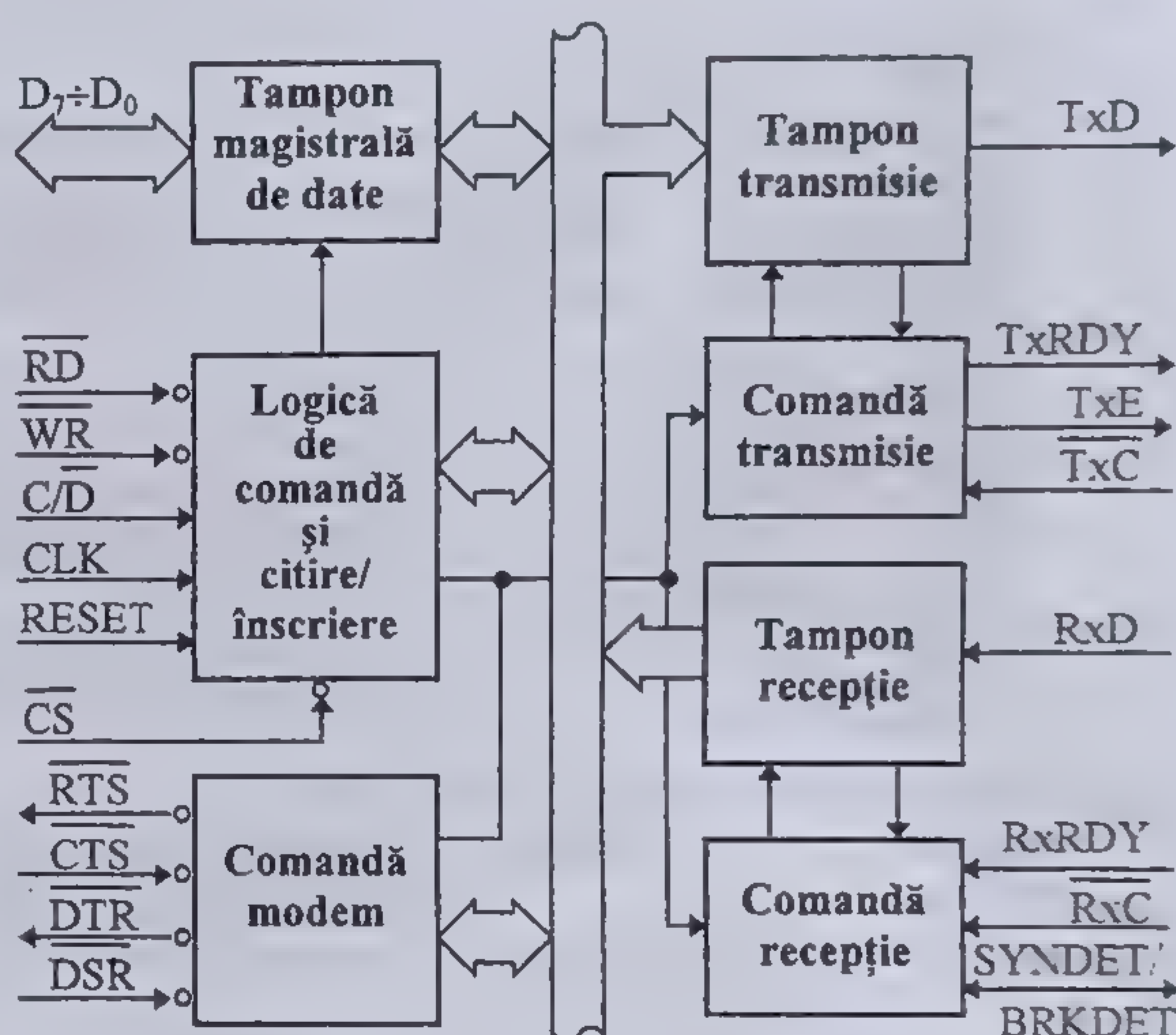


Fig.4.54. Schema bloc a circuitului 8251A

Tab.4.16

\bar{CS}	\bar{RD}	\bar{WR}	C/\bar{D}	Tipul operației efectuate
0	0	1	0	Citire octet din tamponul de recepție
0	1	0	0	Serie octet în tamponul de transmisie
0	0	1	1	Citire cuvânt de stare
0	1	0	1	Înscrisere cuvânt de mod/comandă
0	1	1	x	Nici o operație ($D_7 \div D_0$ în HZ)
1	x	x	x	Nici o operație ($D_7 \div D_0$ în HZ)

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

te, ceea ce permite realizarea unui transfer de tip "full-duplex" pe liniile **TxD** și **RxD**.

Canalul de transmisie este controlat de blocul de comandă aferent și este sincronizat cu semnalul **TxC** (Transmitter Clock). Acest semnal determină ritmul de transmisie a datelor: *pe frontul crescător* determină deplasarea datelor în tamponul paralel/serie și depunerea unui nou bit pe linia **TxD**. În modul sincron, viteza de comunicație (v. §3.1.2.1) este egală cu frecvența semnalului **TxC**, iar în modul asincron poate fi o fracțiune programabilă a frecvenței acestuia (1, 1/16 sau 1/64).

Semnalul **TxRDY** (Transmitter ReaDY) indică microprocesorului că registrul de date al tamponului de transmisie este gol. Acest semnal poate fi utilizat fie ca o cerere de întrerupere, fie pentru testare software, în cadrul unei proceduri de tip interogare. **TxRDY** este resetat pe frontul scăzător al semnalului **WR**, atunci când microprocesorul încarcă un nou cuvânt de date în tamponul de transmisie. Dacă și registrul paralel/serie al tamponului de transmisie este gol, 8251A activează linia **TxEMPTY** (Transmitter EMPTY). La încărcarea unui nou cuvânt în tamponul de transmisie, **TxEMPTY** revine în "0" logic.

Canalul de recepție conține un registru de date, un registru serie/paralel și blocul de comandă care secvențiază recepția serială a datelor. Ritmul de recepție a biților pe linia **RxD** este dat de semnalul **RxC** (Receiver Clock), într-un mod similar cu fixarea ritmului de transmisie prin **TxC**.

Datele seriale sunt eșantionate pe *frontul crescător* al semnalului **RxC** și sunt convertite în format paralel, în registrul de deplasare serie/paralel. La terminarea recepției unui cuvânt, acesta este încărcat în registrul de date al canalului de recepție și se activează linia **RxRDY** (Receiver ReaDY). Se indică astfel microprocesorului existența unui cuvânt recepționat, care poate fi citit. **RxRDY** poate fi utilizat ca semnal de întrerupere, sau poate servi, ca și **TxRDY** sau **TxEMPTY**, într-o procedură de interogare. UCP poate testa valoarea acestor biți, după o operație de citire a stării (Status Read operation).

Semnalul **SYNDET/BRKDET** (SYNc DETect/BReAK DETect) este un semnal de comandă. Mai multe detalii legate de acest semnal vor fi prezentate în paragraful dedicat funcționării circuitului 8251A.

Prin intermediul blocului **comandă modem**, circuitul 8251A oferă posibilitatea de interconectare simplă cu modemuri, necesare în cazul transmisiilor seriale la mare distanță. În acest scop, blocul este prevăzut cu semnale de control/stare corespunzătoare:

DSR (Data Set Ready), de intrare, care indică starea "modem pregătit";

DTR (Data Terminal Ready), de ieșire, care indică modemului starea "terminal de date pregătit";

RTS (Request to Send), de ieșire, utilizat pentru a anunța modemului începerea transmisiei;

CTS (Clear to Send), de intrare, pentru confirmare la cererea de transmisie indicată de **RTS**.

4.8.1.1. Funcționarea circuitului 8251A în modul asincron

După cum s-a arătat în §3.3.1, datele sunt transmise sub forma unor caractere, în care biții de date sunt precedați de un bit de **START**, pe nivel coborât și sunt urmați de un bit de *paritate* (opțional) și de 1, 1½ sau 2 biți de **STOP**, pe nivel logic "1". După resetarea circuitului, linia **TxD** este menținută pe nivel ridicat.

La **transmisie**, 8251A inserează automat bitul de **START** înaintea biților de date și completează caracterul cu bitul de paritate și cu biții de **STOP** (fig.4.55). Șirul de biți astfel obținut este transmis pe ieșirea **TxD**, sub controlul semnalului **TxC**, dacă sunt îndeplinite două condiții:

- dacă s-a activat transmisia, printr-o comandă **TxEN** (Transmit ENable)
- dacă este activ semnalul de confirmare a cererii de transmisie (**CTS** = 0).

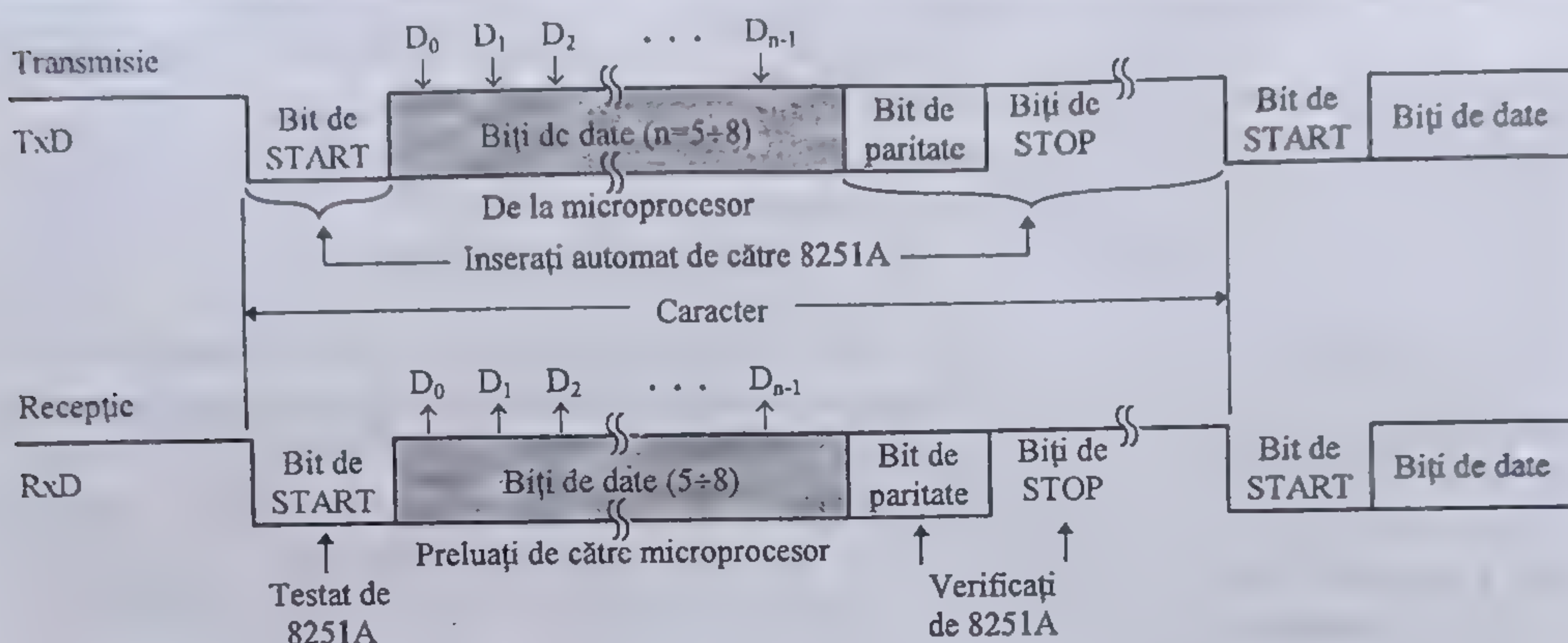


Fig.4.55. Formatul datelor la funcționarea circuitului 8251A în modul asincron

Când registrul de date al transmițătorului este gol, $TxRDY=1$. Dacă nu există nici un caracter de transmis ($TxRDY=TxEMPTY=1$), atunci ieșirea TxD rămâne pe nivel ridicat. Linia TxD poate fi forțată în orice moment în "0" logic, printr-o comandă SBRK (Send BReak character), situație în care o eventuală transmisie în curs este abandonată.

Recepția datelor pe linia RxD este activată printr-o comandă RxEN (Receive ENable). În lipsa unui caracter, linia RxD se află pe nivel ridicat. Un front căzător pe această linie este interpretat ca începutul unui bit de START. Pentru a filtra impulsurile parazite, 8251 testează din nou starea acestui bit la jumătatea perioadei de recepție, dar numai în cazul în care aceasta este egală cu 16 sau 64 de perioade ale semnalului RxC . Dacă linia RxD este găsită în "1" logic, recepția este abandonată și circuitul rămâne în așteptarea unui nou caracter. Dacă testarea validează existența unui bit de START (RxD în "0" logic), 8251A continuă cu recepția biților de date, bitul de paritate (dacă a fost programat) și bitul sau biții de STOP. Biții de date recepționați sunt convertiți în format paralel și transferați în registrul de date, pentru a fi preluați de către microprocesor. Existența datei valide în tamponul de recepție este semnalată de către 8251A prin activarea semnalului RxRDY.

La recepție, dacă este detectată o *eroare de paritate*, se setează bistabilul PE (Parity Error) din registrul de stare. Dacă se detectează un nivel coborât la verificarea biților de STOP, 8251A setează un bistabil de *eroare de încadrare* (FE - Framing Error). În cazul în care microprocesorul întârzie să preia data din registrul de date al receptorului, următorul caracter recepționat se încarcă peste cel precedent și se setează un bistabil de *eroare de suprapunere* (OE - Overrun Error). Caracterul necitit la timp de către UCP se pierde. Nici unul din biții de eroare menționați nu oprește funcționarea circuitului 8251, dar microprocesorul poate afla despre apariția acestor erori prin citirea registrului de stare. Indicatorii de eroare pot fi reșetați, simultan, printr-o comandă ER (Error Reset). La detectarea unui caracter BREAK, linia BRKDET trece în "1" logic și este setat bistabilul cu același nume din registrul de stare.

4.8.1.2. Modul de lucru sincron

Conform celor prezentate în §3.3.2, funcționarea circuitului 8251A în modul sincron se caracterizează printr-o sincronizare la nivel de bloc de date și nu la nivel de caracter, ca în cazul modului asincron. Astfel, un bloc de date este format dintr-un șir succesiv de biți, fără biți de delimitare între cuvinte, precedat de unul sau două caractere de sincronizare succesive (SYNC).

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

Circuitul nu dispune de un bloc propriu de calcul al restului CRC, deci aceste informații de control trebuie calculate și transmise tot de către microprocesor, la sfârșitul fiecărui bloc de date.

La transmisie (fig.4.56), ieșirea TxD se menține continuu pe nivel ridicat, până când UCP inițiază transmisia unui bloc de date prin înscrierea în tamponul de transmisie a caracterelor de sincronizare.

Ca și în cazul modului asincron, dacă $TxEN=1$ și $\overline{CTS}=0$, începe transmisia serială a biților caracterelor SYNC, urmați de biții de date, pe frontul căzător al semnalului \overline{TxC} . Dacă la activarea liniei TxRDY microprocesorul nu înscrie noi date în tamponul de transmisie înainte ca acesta să se golească, circuitul 8251A va insera automat caractere SYNC (1 sau 2, după cum a fost programat) în fluxul de date. În același timp, se va activa semnalul TxEMPTY, care va fi resetat în momentul în care se va înscrie un nou cuvânt de date.

Recepția în modul sincron este de două tipuri: cu *sincronizare internă* sau *externă*.

În *modul cu sincronizare internă*, inițierea recepției este comandată prin trecerea circuitului 8251A în modul de "căutare" a caracterelor SYNC (HUNT Mode). Acest lucru se realizează printr-o comandă EH (Enter HUNT Mode). Biții citiți de pe linia RxD, pe frontul crescător al semnalului \overline{RxC} , sunt comparați cu biții caracterului (caracterelor) SYNC, stabiliți la programarea modului sincron. În momentul în care ultimii 8 (sau 16 biți) citiți coincid cu șirul de 8 (sau 16) biți de sincronizare, circuitul 8251A iese din modul HUNT. Obținerea sincronizării este semnalizată prin activarea semnalului SYNDET și setarea bitului cu același nume din registrul de stare. La citirea cuvântului de stare, se realizează și dezactivarea liniei SYNDET.

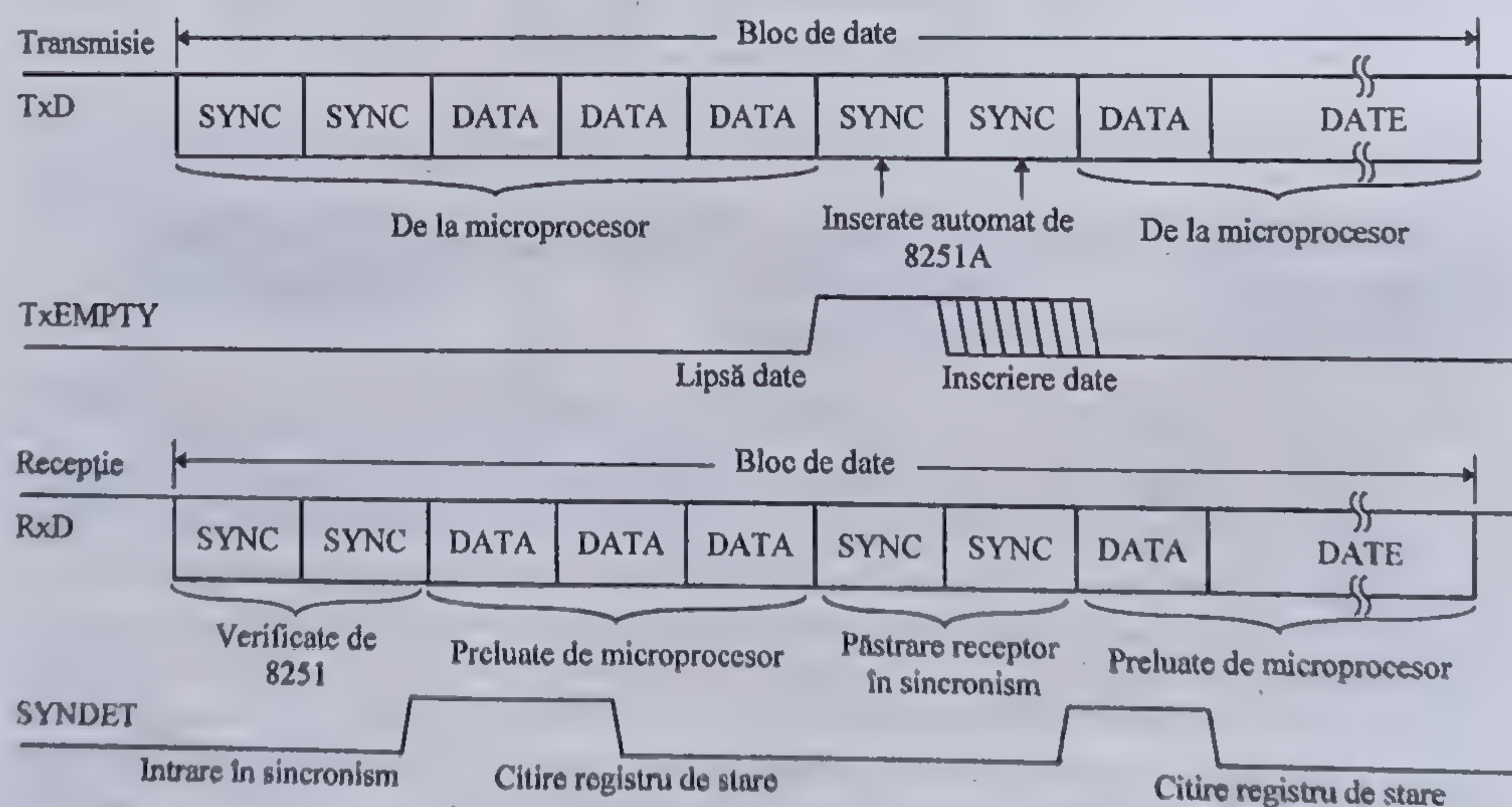


Fig.4.56. Funcționarea circuitului 8251A în modul sincron (cu sincronizare internă)

Caracterele de sincronizare recepționate sunt eliminate din blocul de date, ele nefiind preluate de către microprocesor. După realizarea sincronizării, începe recepționarea blocului de date. În cazul transmisiei sincrone se poate testa numai apariția erorilor de paritate și de suprapunere. La pierderea sincronismului, UCP poate cere receptorului să reîntre în modul "căutare".

În *modul cu sincronizare externă*, singură deosebire este aceea că inițierea recepției se face printr-o activare a liniei SYNDET, care este acum linie de intrare. Astfel, un semnal "1" logic pe această linie, furnizat de logica externă de sincronizare, forțează ieșirea circuitului 8251A din

modul HUNT și începerea asamblării biților de date, odată cu primul front crescător pe linia \overline{RxC} . În acest mod, detectarea internă a caracterelor de sincronizare este dezactivată. Linia SYNDTET poate reveni în "0" logic după o perioadă a semnalului \overline{RxC} .

4.8.1.3. Programarea circuitului 8251A

După resetare, circuitul 8251A trebuie programat în unul din cele două moduri descrise în paragraful anterior. Aceasta constă din înscrierea, în registrul de mod/comandă, a două tipuri de cuvinte: *de mod* (Mode Instruction) și respectiv *de comandă* (Command Instruction).

Prin *cuvântul de mod* se programează modul de lucru și se definesc parametrii comunicației seriale, corespunzător modului de lucru selectat: *asincron* (AMW), respectiv *sincron* (SMW).

În cazul transferului asincron, biții B2 și B1 stabilesc valoarea factorului de viteză, k_v , prin relațiile:

$$f_{TxC} = k_v \cdot V_T [\text{Hz}] ; \quad f_{RxC} = k_v \cdot V_R [\text{Hz}] ,$$

D ₇		D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	AMW
S2	S1	EP	PEN	L2	L1	B2	B1		
Număr biți de STOP		Tip paritate	Validare generare/control paritate	Număr biți de date		Factor de viteză (mod asincron)			
0 0 - inactiv		0 - impară 1 - pară	0 - fără paritate 1 - cu paritate	0 0 - 5 biți		0 1 - 1			
0 1 - 1 bit				0 1 - 6 biți		1 0 - 16			
1 0 - 1½ biți				1 0 - 7 biți		1 1 - 64			
1 1 - 2 biți				1 1 - 8 biți					

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	
SCS	ESD	EP	PEN	L2	L1	0	0	SMW
Număr caractere SYNC	Tip sincro- nizare	Tip paritate	Validare generare/control paritate	Număr biți de date		Mod sincron (Factor de viteză 1)		
0 - două 1 - unu	0 - internă 1 - externă	0 - impară 1 - pară	0 - fără paritate 1 - cu paritate	0 0 - 5 biți				
				0 1 - 6 biți				
				1 0 - 7 biți				
				1 1 - 8 biți				

unde f_{TxC} și f_{RxC} sunt frecvențele semnalelor de tact de transmisie (\overline{TxC}), respectiv de recepție (\overline{RxC}), iar V_T și V_R sunt vitezele de transmisie și respectiv de recepție, în [Baud]. Un factor $k_v=1$ se poate utiliza numai dacă transmițătorul și receptorul lucrează cu același semnal de tact. Dacă se utilizează semnale de tact independente, pentru o sincronizare corectă a transferului serial trebuie utilizat neapărat un factor $k_v = 16$ sau 64. Cunoscând viteza de comunicație (de obicei $V_T=V_R$) și modul de generare a semnalului de tact, se poate calcula frecvența acestuia.

Spre exemplu, dacă se dorește realizarea unui transfer cu $V_C = 9600\text{Baud}$, folosind semnale de tact independente, pentru $k_v = 16$ ($B2\ B1 = 10$ în AMW) sau 64 ($B2\ B1 = 11$), rezultă $f_{TxC/RxC} = 153,6\text{KHz}$, respectiv $614,4\text{KHz}$. În cazul transferului sincron, acest factor este întotdeauna 1, deci transmițătorul și receptorul trebuie să lucreze cu un semnal de tact comun.

La ambele moduri de transfer, prin biții L2 și L1 poate fi programată lungimea caracterului transmis, mai exact numărul de biți de date dintr-un caracter, iar prin biții EP și PEN se poate stabili dacă la transmisie se va genera, iar la recepție se va verifica bitul de paritate, respectiv tipul de paritate utilizat.

Biții D₇ și D₆ stabilesc modul de încadrare a cuvântului de date, care este specific fiecărui tip de comunicație: la modul asincron - cu biți de STOP (S2 S1), iar la cel sincron - cu caractere

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

SYNC (D_7 =SCS - Single Character Sync). În plus, la modul sincron se poate alege și tipul de sincronizare, internă sau externă, prin bitul D_6 =ESD (External Sync Detect).

Funcționarea circuitului poate fi controlată în orice moment, de către microprocesor, prin cuvinte de comandă (CW) înscrise în registrul de mod/comandă.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	CW
EH	IR	RTS	ER	SBRK	RxEN	DTR	TxEN	
Intrare în mod "HUNT"	Resetare internă	Forțează $\overline{\text{RTS}}=0$ (pin)	Resetare bistabile de eroare	Forțează $\text{TxD}=0$ (pin)	Validare recepție	Forțează $\overline{\text{DTR}}=0$ (pin)	Validare transmisie	

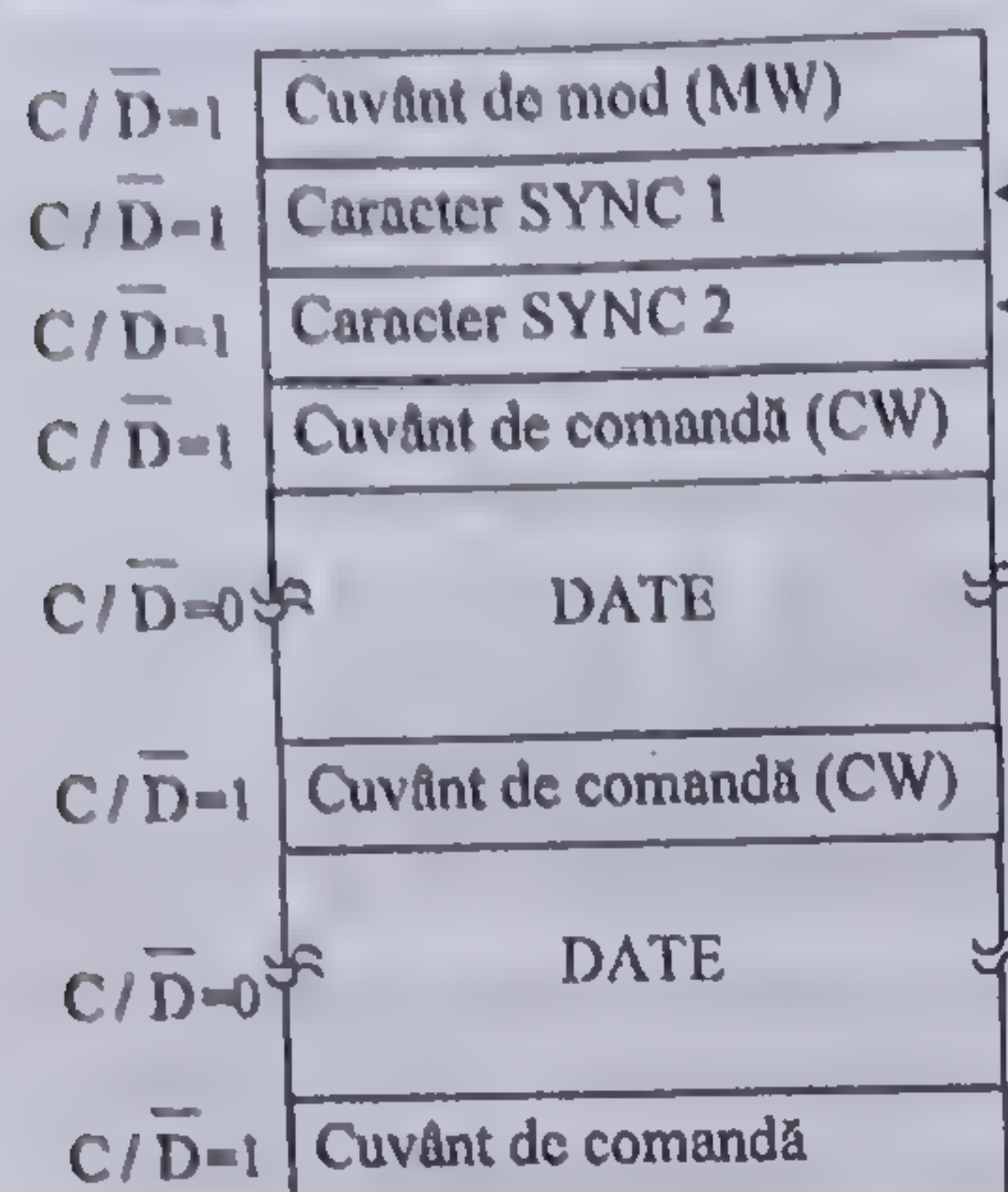
Fiecare bit este activ pe "1" logic și reprezintă o comandă din cele prezentate deja, la funcționarea circuitului. Biții D_0 =TxEN și D_2 =RxEN permit activarea/dezactivarea transmisiei, respectiv a recepției. Biții D_5 =RTS și D_1 =DTR comandă starea pinilor cu același nume (care sunt activi pe nivel coborât) și permit astfel microprocesorului să informeze modemul despre starea în care se află interfața serială (cerere de transmisie, terminal de date pregătit). Întreruperea transferului unui mesaj poate fi anunțată prin forțarea continuu pe "0" logic a liniei TxD, printr-o comandă cu D_3 =SBRK=1 (Send BRaK character). Bistabilele de eroare din registrul de stare pot fi șterse printr-un cuvânt de comandă cu D_4 =ER=1. Resetarea circuitului poate fi realizată și prin program, printr-un cuvânt de comandă cu D_6 =IR=1 (Internal Reset), având același efect cu activarea semnalului RESET. Printr-o astfel de comandă, microprocesorul poate readuce 8251A în starea "idle", care trebuie să fie urmată de o reprogramare a parametrilor de lucru ai circuitului (un nou cuvânt de mod). Bitul D_7 =EH=1 determină intrarea circuitului în modul "căutare caracter(e) de sincronizare" (Enter Hunt mode), având efect numai în modul sincron, cu sincronizare internă.

Starea canalelor de transmisie și de recepție poate fi cunoscută de microprocesor prin citirea unui registru de stare; *cuvântul de stare* (SW) are următoarea structură:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	SW
DSR	SYNDET/BRKDET	FE	OE	PE	TxEMPTY	RxRDY	TxRDY	
Îndică dacă $\overline{\text{DSR}}=0$ (pin)		Eroare de încadrare	Eroare de suprapunere	Eroare de paritate				Reflectă întocmai starea pinilor cu același nume

Programarea circuitului 8251A necesită o succesiune bine definită de operații de înscriere cuvinte de mod și comandă ($C/\overline{D}=1$), urmată de transferul de date ($C/\overline{D}=0$), de citirea registrului de stare ($C/\overline{D}=1$) pentru verificarea terminării transferului și de noi cuvinte de comandă, așa cum se arată în fig.4.57a.

Conform specificațiilor firmei producătoare [43], pentru ca circuitul să intre cu certitudine în starea de așteptare a cuvântului de mod, după resetarea HW de după alimentare trebuie să urmeze mai întâi o *secvență de sincronizare* specifică (fig. 4.57b). Această secvență constă din transmiterea succesivă a trei octeți cu valoarea 00h în registrul de mod/comandă, urmați fiind de o comandă de resetare software (IR=1 în CW). În plus, este necesar să se respecte timpul de refacere al circuitului între două înscrieri succesive (t_{RV} - Recovery Time), care trebuie să fie de minimum 16 perioade ale semnalului de tact CLK. După executarea secvenței de sincronizare, se poate înscrie cuvântul de mod, urmat de un cuvânt de comandă - pentru modul asincron, respectiv de 1 sau 2 caractere SYNC și un cuvânt de comandă - în modul sincron. Și în acest caz, între două cuvinte succesive înscrise în registrul de mod/comandă trebuie respectat același timp de refacere, t_{RV} .



Numai în
modul sincron
(după SMW)

b)

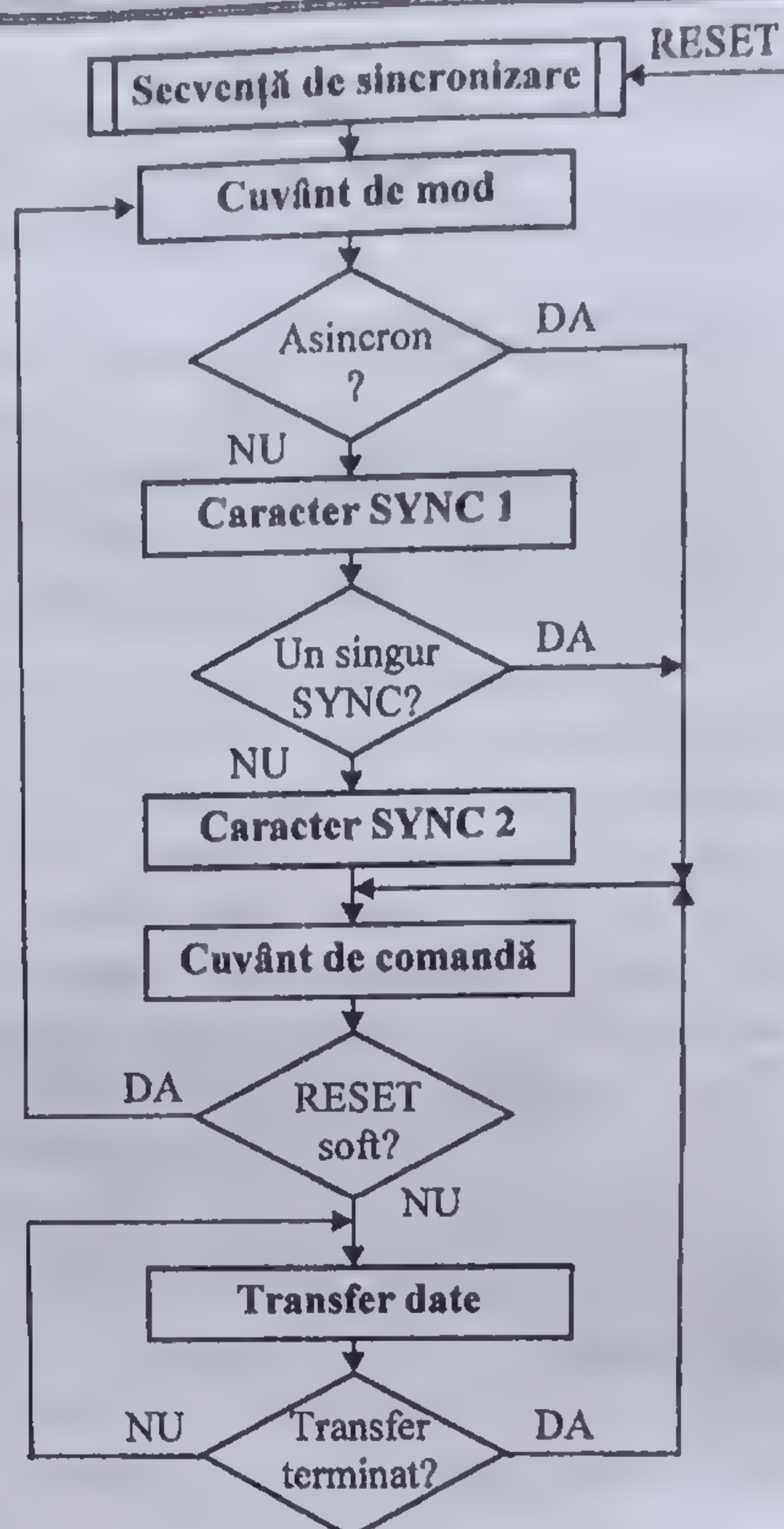


Fig.4.57. Succesiunea operațiilor (a) și schema logică (b) la programarea circuitului 8251A

În urma efectuării acestor operații, circuitul 8251A este inițializat și apt să transfere date, în modul programat. La înscrierea datelor în tamponul de transmisie nu este necesar să se țină cont de tipul de refacere, timpul necesar pentru transferul serial al unui caracter fiind mult mai mare decât t_{RV} .

4.8.2. Utilizarea circuitului USART 8251A

Programarea interfeței seriale 8251A se va exemplifica pentru cazul comunicației seriale asincrone, între două sisteme cu microprocesoare de 8 biți, fără modem (fig.4.58).

Se utilizează adaptoare de comunicație în standardul RS232 (v. §3.2.1), de tipul MC1488 la emisie (alimentate la $\pm 12V$) și MC1489 la recepție (alimentate la $+5V$). Nu se utilizează protocolul RTS/CTS, liniile \overline{RTS} și \overline{CTS} fiind interconectate în buclă locală, pentru validarea automată a transmisiei, de îndată ce apare o cerere în acest sens ($\overline{RTS}=0 \Rightarrow \overline{CTS}=0$). Tactul de sincronizare transmisie-recepție este generat de sistemul de introducere a timpului (SIT) al fiecărui sistem în parte, realizat - de exemplu - cu circuite PIT8253 (v. §4.5.2).

Tratarea semnalelor RxRDY (recepție caracter) și respectiv TxRDY sau TxEMPTY (transmisie caracter) se realizează în sistemul de întreruperi (SINT) local. Funcționarea fiecărui circuit 8251A este sincronizată cu propria unitate centrală, folosind drept semnale de tact Φ_{2TTL} la 8080, respectiv CLK OUT la 8085. Resetarea HW a circuitelor 8251A se face odată cu activarea

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

semnalelor RESET ale celor două sisteme. Linia C/\overline{D} se conectează la linia A_0 de pe magistrala de adrese, iar linia \overline{CS} este controlată de logica de decodificare și selecție porturi.

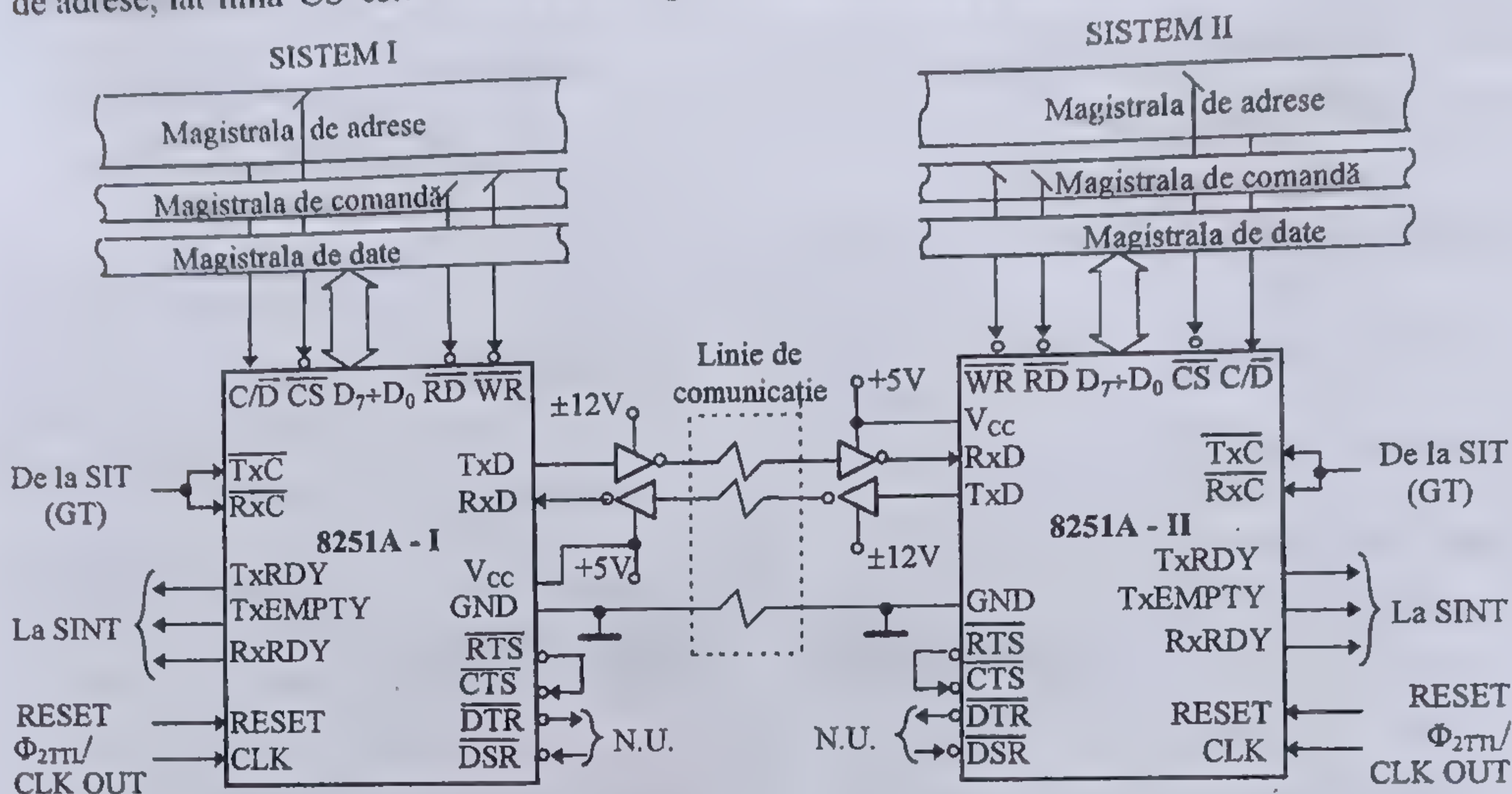


Fig. 4.58. Sisteme cu microprocesoare de 8 biți, interconectate serial prin circuite 8251A

; SECVENȚA DE SINCRONIZARE A CIRCUITULUI 8251A (I sau II)

USART	EQU	80h	; Adresa de bază 8251A - I
RESOFT	EQU	01000000b	; Comandă resetare internă
MVI	B,03h		; Se vor înscrie 3 octeți,
XRA	A		; cu valoarea 00h (A=00h),
sincro: OUT	USART+1		; (10t _{cy}) în registrul de mod/comandă (cu adresa 81h),
CALL	delay		; (cel puțin 17+10=27t _{cy})
DCR	B		; (5 sau 4t _{cy})
JNZ	sincro		; (10t _{cy}) succesiv, cu asigurarea t _{RV} > 16t _{cy} .
MVI	A,40h		; Încarcă în acumulator comanda de resetare.
OUT	USART+1		; Resetare internă 8251A
CALL	delay		

Subrutina de întârziere (delay) asigură o margine de siguranță pentru timpul de refacere, t_{RV}, pentru situația cea mai dezavantajoasă. Spre exemplu, chiar dacă rutina va conține numai instrucțiunea de revenire, combinația instrucțiunilor CALL delay (17t_{cy}) și RET (10t_{cy}) vor asigura t_{RV} >> 16t_{cy}.

; PROGRAMARE ÎN MODUL ASINCRON

AMW	EQU	01111110b	; Cuvântul de mod asincron: 8 biți de date pe caracter,
			; paritate pară, 1 bit de STOP și factor de viteză k _v =16.
MVI	A,AMW		; Încarcă în acumulator cuvântul de mod.
OUT	USART+1		; Înscrie cuvântul de mod în registrul de mod/comandă (81h).
CALL	delay		

; COMANDĂ ACTIVAREA TRANSMISIEI ȘI A RECEPȚIEI

MVI	A,00100101b	; TxEN=1, RTS=1, RxEN=1.
OUT	USART+1	; Validează transmitia și recepția serială a datelor.

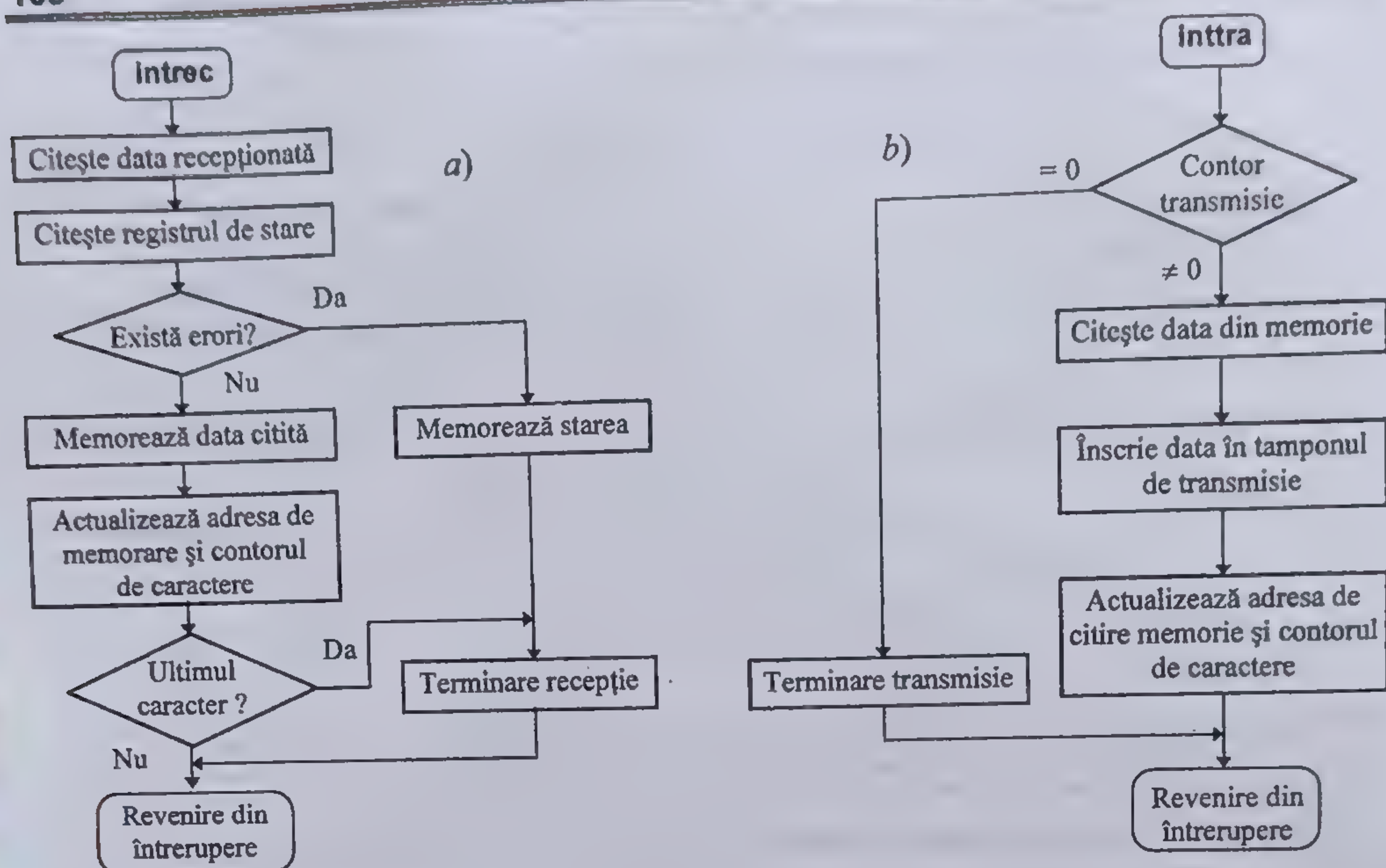


Fig.4.59. Schema logică a rutinelor de întrerupere la recepție (a) și transmisie (b)

În momentul recepției unui caracter, se activează linia RxRDY și se lansează o cerere de întrerupere. Rutina de tratare a întreruperii de recepție, *intrec* (fig.4.59a), trebuie să preia data din tamponul de recepție, apoi să testeze starea bistabilelor de eroare. Dacă nu există erori la recepție, data citită este depusă într-o zonă de memorie tampon, pentru a fi preluată ulterior de către o altă rutină. Dacă a apărut o eroare la recepție, data citită este ignorată și se iau măsuri pentru semnalizarea erorii, în vederea repetării transferului (v. §3.1.4.1).

După transmisia unui caracter sau la golirea tamponului de transmisie se lansează o cerere de întrerupere ca urmare a activării uneia din liniile TxRDY sau TxEMPTY. Rutina de tratare a întreruperii de transmisie generată de TxRDY, *intră* (fig.4.59b), verifică mai întâi dacă mai sunt caractere de transmis. Dacă da, citește următorul octet din memorie, îl înscrie în registrul de date al tamponului de transmisie, actualizează contorul de caractere rămase de transmis și adresa de citire din memorie, revenind din rutină. Dacă nu mai sunt date de transmis, se semnalizează terminarea transmisiei mesajului curent și se revine din întrerupere, fără a mai transmite un nou caracter.

O posibilă implementare prin program a schemelor logice din fig.4.59 este prezentată în cele ce urmează. Rutina de întrerupere la recepție asigură preluarea unui mesaj de maximum 256 de octeți, în zona de memorie rezervată la adresa *bufrec*. În acest scop se utilizează un contor de octeți (*cntrec*) rămași de recepționat și un pointer de date (*ptrrec*), care conține adresa următoarei locații libere din zona de memorie rezervată.

Rutina de întrerupere la transmisie preia următorul caracter din zona de memorie de 256 de octeți, *buftra*, de la adresa dată de pointerul de transmisie *ptrtra*, atât timp cât contorul de octeți rămași de transmis, *cnttra*, este nenul.

Terminarea operațiilor de recepție/transmisie mesaj este semnalizată prin poziționarea indicatorilor *recmes* și respectiv *trames*, pentru care s-a rezervat câte un octet în segmentul de date.

Capitolul 4 - Sisteme cu microprocesoare Intel de 8 biți

În cazul modului sincron de operare, programarea circuitului 8251A decurge astfel:

- 1) Se transmite o secvență de sincronizare, similară cu cea descrisă la modul asincron.
- 2) Se înscrie cuvântul de mod sincron (SMW).
- 3) Se înscriu unul sau două caractere SYNC, tot în registrul de mod/comandă (cu $C/\overline{D}=1$).
- 4) Se activează transmisia ($TxEN=1$, $RTS=1$), recepția ($RxEN=1$) și intrarea circuitului în modul "căutare caractere de sincronizare" ($EH=1$).

; RUTINA DE ÎNTRERUPERE LA RECEPȚIE

intrec:

```

PUSH PSW      ; Salvare regiștri
PUSH B        ; în stivă.
PUSH H
IN  USART     ; Citește data
                ; (C /  $\overline{D}$ =0).
MOV  B,A      ; Memorează în B.
IN  USART+1   ; Citește registrul de
                ; stare (C /  $\overline{D}$ =1).
ANI  00111000b
JZ   noerr    ; Caracter eronat.
STA  coderr   ; Memorează starea.
JMP  endrec

```

noerr:

```

MOV  A,B      ; Caracter corect.
LHLD ptrrec   ; HL=adr. curentă.
MOV  M,A      ; Memorează data.
INX  H
SHLD ptrrec   ; Actualiz. adresa.
LDA  cntrec   ; Decrem. Contorul.
DCR  A
STA  cntrec   ; Actualiz. Contorul.
JNZ  revinr   ; Ultimul caracter?

```

endrec:

```

MVI  A,10h    ; Resetare erori și
OUT  USART+1   ; invalidare recepție
                ; (C /  $\overline{D}$ =1).
MVI  A,01h    ; Indică terminarea
STA  recmes    ; recepției mesajului.

```

revinr:

```

POP  H        ; Reface registrele
POP  B        ; de pe stivă.
POP  PSW
EI          ; Validare într.
RET         ; Revenire.

```

DSEG ; Segmentul de date.

```

bufrec: DS 256 ; Mem. tampon recepție.
ptrrec: DS 2   ; Adresa curentă (pointer).
cntrec: DS 1   ; Contor recepție.
coderr: DS 1   ; Cod eroare recepție.
recmes: DS 1   ; Indicator recepție mesaj.

```

; RUTINA DE ÎNTRERUPERE LA TRANSMISIE

inttra:

```

PUSH PSW      ; Salvare regiștri
PUSH H
LDA  cnttra   ; Contorul = 0?
ORA  A
JZ   endtra   ; Da, deci nu mai sunt
                ; caractere de transmis.
DCR  A        ; Nu, deci actualizează.
STA  cnttra
LHLD ptrtra   ; HL=adresa curentă.
MOV  A,M      ; Citește data.
INX  H
SHLD ptrtra   ; Actualizează adresa.
OUT  USART    ; Transmite data (C /  $\overline{D}$ =0)
JMP  revint

```

endtra:

```

MVI  A,0      ; Forțează  $\overline{RTS}=1$ :
OUT  USART+1   ; (terminare transmisie)
                ; (C /  $\overline{D}$ =1).
MVI  A,01h    ; Indică terminarea
STA  trames    ; transmisiei mesajului.

```

revint:

```

POP  H        ; Reface registrele
POP  PSW
EI          ; Validare întreruperi.
RET         ; Revenire.
DSEG        ; Segmentul de date.

```

```

buftra: DS 256 ; Mem. tampon transmisie.
ptrtra: DS 2   ; Adresa curentă (pointer).
cnttra: DS 1   ; Contor transmisie.
trames: DS 1   ; Indicator transmisie mesaj.

```

Rutinele de transmisie/recepție mesaj corespund protocolului de comunicație de pe nivelul legăturii de date. Acestea asigură formarea pachetelor de date prin încadrarea biților între delimitatori de câmp (v. §3.3.2 și §3.3.3) și controlul corectitudinii transferului serial, folosind coduri ciclice redundante (v. §3.1.4.1).

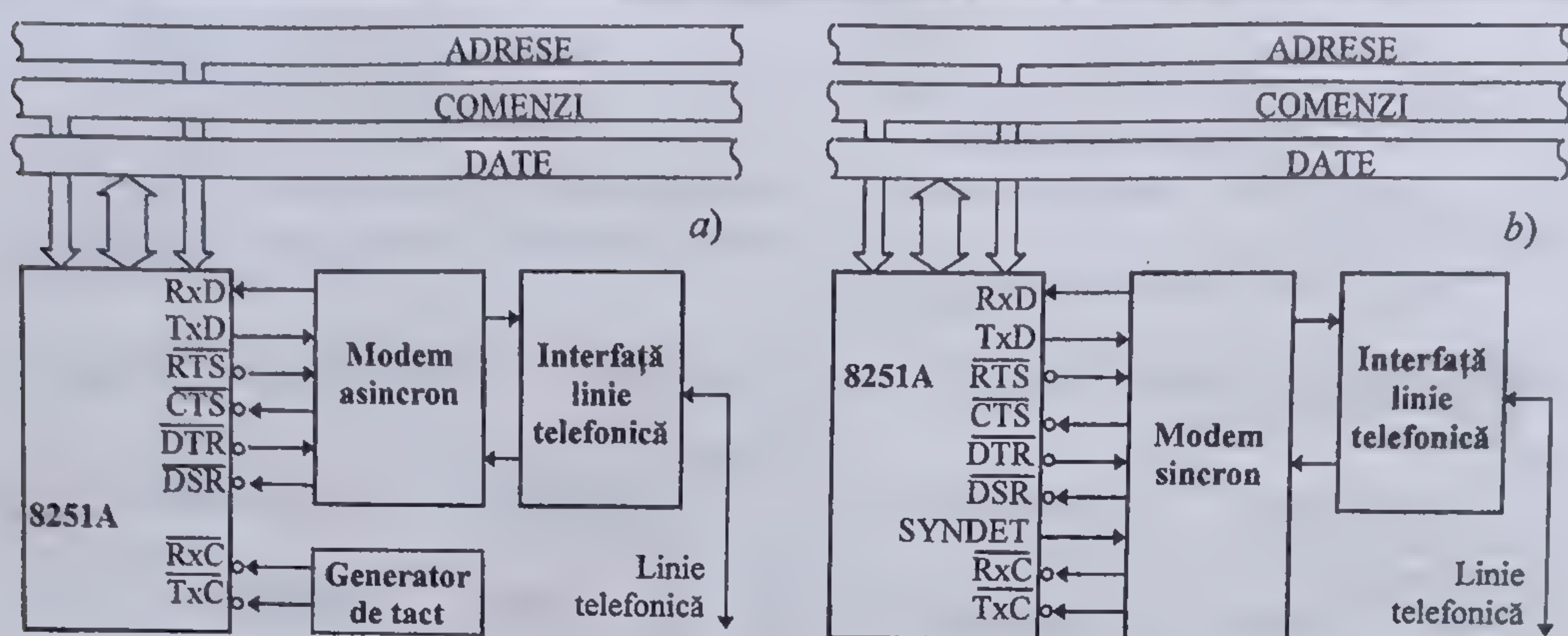


Fig.4.60. Interfațarea cu modemuri (a) asincrone și (b) sincrone

În fig.4.60 sunt prezentate două exemple de interfațare a circuitului 8251A cu linia telefonică, folosind modemuri asincrone (a), respectiv sincrone (b).

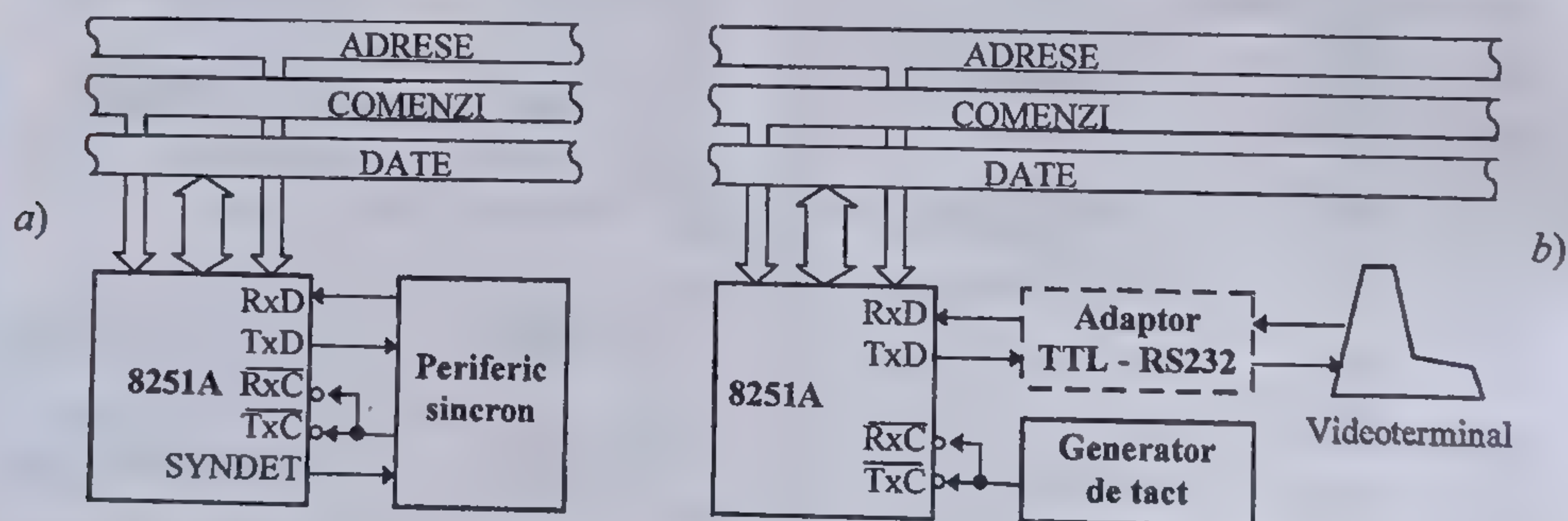


Fig.4.61. Interfațarea directă cu un periferic sincron (a), respectiv cu un videoterminal (CRT) (b)

De asemenea, USART 8251A se poate interfața și direct, fără modem, atât cu periferice sincrone (v.fig.4.61a), cât și cu videoterminale inteligente (de obicei în modul asincron), așa cum se arată în fig.4.61b. De obicei, videoterminalul este prevăzut cu o interfață RS232, ceea ce necesită o adaptare suplimentară TTL - RS232. Sincronizarea comunicației poate fi realizată cu un generator de tact independent sau pilotat cu semnalul de tact al sistemului cu microprocesor.



Sisteme cu microprocesorul Z80

Microprocesorul Z80, ca și 8085A, este o unitate centrală de procesare într-un singur cip, integrând circa 8500 tranzistoare. Realizat de principalii trei proiectanți ai microprocesorului 8080A, produsul firmei Zilog asigură atât compatibilitatea la nivel de cod cu acesta, cât și noi facilități, neîntâlnite la microprocesoarele de 8 biți ale firmei Intel. Ca urmare, sistemele cu Z80 reprezintă una din soluțiile avantajoase pentru realizarea structurilor de conducere automată bazate pe microprocesoare de uz general.

5.1. Microprocesorul Z80

Z80 a fost produs inițial în patru variante NMOS: Z80 (2,5MHz), Z80A (4MHz), Z80B (6MHz) și Z80L (Low power - la 1, 1,5 sau 2,5MHz) [44]. În prezent, firma Zilog pune la dispoziția proiectanților variantele Z8040004 (4MHz), Z8040006 (6,17MHz), Z8040008 (8MHz), precum și alte patru variante CMOS, mult mai performante: Z84C0006 (6,17MHz), Z84C0008 (8MHz), Z84C0010 (10MHz), Z84C0020 (20MHz), care permit o adaptare facilă la diferite condiții impuse de aplicație [44]. Toate aceste variante sunt complet compatibile în ceea ce privește arhitectura internă, semnalele la pini și setul de instrucțiuni.

Principalele caracteristici ale familiei Z80, în comparație cu produsele Intel, sunt rezumate în continuare:

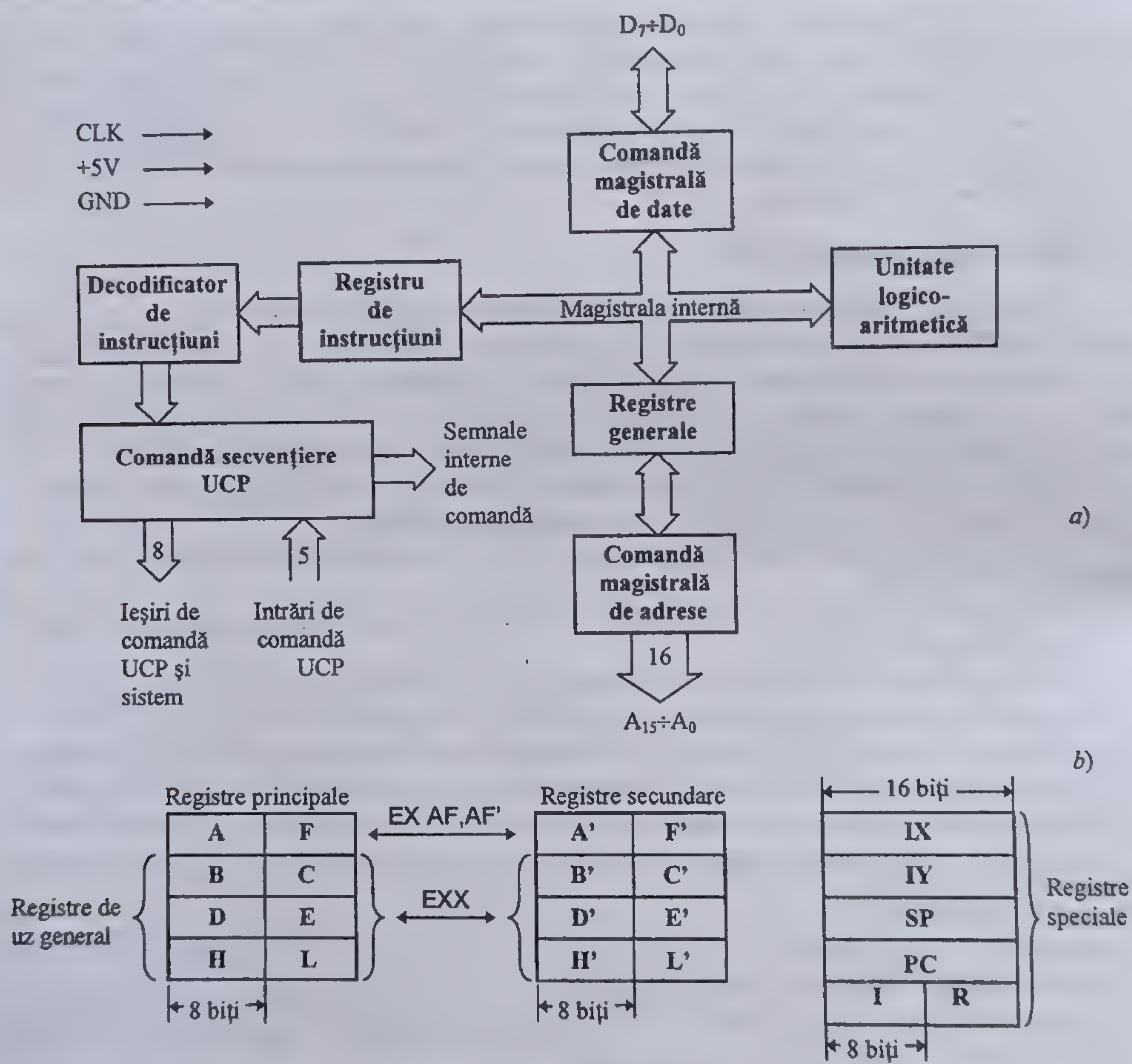
- set extins de instrucțiuni, format din cele 78 ale procesorului 8080A, la care se adaugă 80 de instrucțiuni proprii;
- noile instrucțiuni includ operații suplimentare pe 4, 8 și 16 biți, precum și la nivel de bit;
- un număr crescut de moduri de adresare, cuprinzând adresarea indexată, relativă și pe bit;
- duplicarea registrelor de uz general și a registrelor cu indicatorii de condiții, fapt ce simplifică considerabil comutarea contextului în programarea cu subrutine și în servirea întreruperilor;
- două registre index, de 16 biți, care facilitează procesarea tabelor și a structurilor de date;
- trei moduri vectorizate de tratare a întreruperilor mascabile (pe linia $\overline{\text{INT}}$) și o întrerupere nemascabilă ($\overline{\text{NMI}}$);
- durate tipice de 1,6μs (la Z80), 1μs (la Z80A) și 0,66μs (la Z80B) pentru execuția unei instrucțiuni, la frecvența maximă de lucru;
- un semnal de tact cu o singură fază, simetric și de nivel TTL, fără o limită inferioară a frecvenței de lucru (până la c.c., pentru primele variante NMOS și pentru variantele CMOS), ceea ce la procesoarele Intel nu este posibil;
- o singură tensiune de alimentare, de +5V;
- semnalele la pini sunt compatibile TTL, ieșirile având un fan-out de 1 sarcină TTL ($I_{OL}=1,8\text{mA}$ - variantele NMOS, $I_{OL}=2\text{mA}$ - variantele CMOS).

Mai trebuie menționat și faptul că firma Zilog furnizează o întreagă familie de dispozitive specializate, direct interfațabile cu microprocesorul Z80: PIO-Z80 (Parallel Input/Output), CTC-Z80 (Counter/Timer Circuit), DART-Z80 (Dual Asynchronous Receiver/Transmitter), SIO-Z80 (Serial Input/Output) și DMAC-Z80 (Direct Memory Access Controller), deosebit de performante, care beneficiază și de avantajul unui număr sporit de instrucțiuni I/E. De asemenea, în prezent firma Zilog oferă diferite dispozitive I/E împachetate pe același cip, atât fără UCP (KIO-Z80 -

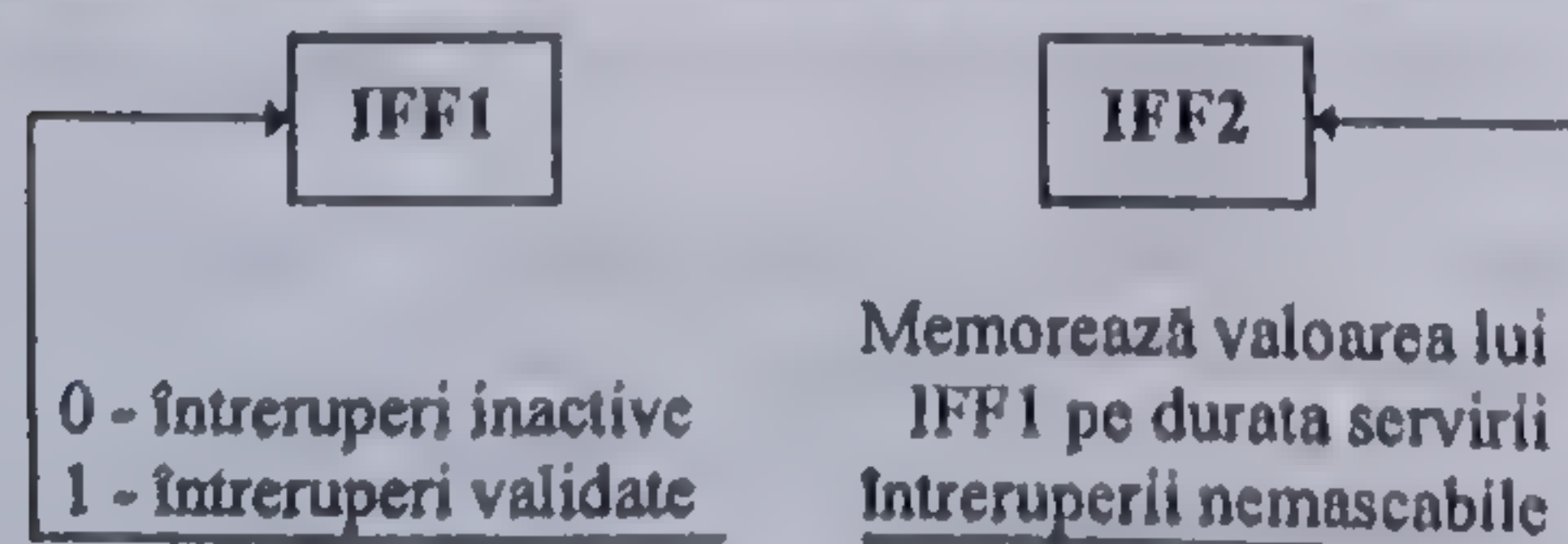
Serial/Parallel/Counter/Timer), precum și împreună cu UCP-Z80 (Z84015/Z84C15 - Intelligent Peripheral Controller) [45]. Ultima variantă face tranziția către microcontrolerele Z8 [46].

Toate aceste facilități permit dezvoltarea unei aplicații într-un timp mai scurt și reducerea substanțială a spațiului de memorie ocupată (cu 25+50%), față de un sistem realizat cu 8080/8085.

Schema bloc a microprocesorului Z80 este prezentată în fig.5.1a. Acesta este organizat în jurul unei magistrale interne de 8 biți și are ca elemente principale o unitate logico-aritmetică, un



Bistabile de stare a sistemului de întreruperi mascabile



Bistabile de mod pentru întreruperile mascabile

IMF ₀	IMF ₁	
0	0	- Modul 0 - compatibil 8080
0	1	- Neutilizat
1	0	- Modul 1 - pentru periferice non-Z80
1	1	- Modul 2 - specific familiei Z80

Fig.5.1. Schema bloc (a) și registrele interne accesibile utilizatorului (b) la microprocesorul Z80

bloc de registre, circuite de comandă și generare a magistralelor de date și adrese, registrul de instrucțiuni împreună cu blocul de decodificare, precum și blocul de comandă a secvențierii

Partea a II-a - Sisteme cu microprocesoare de 8 biți

operațiilor interne. Microprocesorul dispune de un număr de 13 semnale de sistem și de comandă a UCP, care vor fi descrise ulterior.

Caracteristic arhitecturii interne a microprocesorului Z80 este prezența a două seturi de registre accesibile programatorului (fig.5.1b): un set principal (main register set), identic în componență și funcționalitate cu cel de la procesoarele Intel și un set secundar (alternate register set), utilizat numai pentru memorarea temporară a conținutului setului principal, prin instrucțiuni dedicate (EX AF, AF' și EXX). Practic, setul secundar de registre formează o mică stivă RAM internă, care favorizează implementarea tehnicilor de programare de tip "foreground-background" și permit un răspuns rapid la întreruperi.

Spre deosebire de procesoarele Intel, registrul cu indicatorii de condiție este notat la Z80 cu F (F') și este format din 6 bistabile, poziționate ca în fig.5.2.

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
S	Z	-	H	-	P/V	N	C

Fig.5.2. Configurația registrului F (F')

Atât funcționalitatea, cât și poziția indicatorilor S, Z, H, P/V și C este identică cu cea de la procesoarele Intel (v. §4.1.1). Indicatorul de transport auxiliar, AC, utilizat la efectuarea

calculelor în aritmetica BCD, este denumit la Z80 indicator de semi-transport (H - Half carry). În plus, tot pentru operarea în BCD, s-a introdus un fanion ce indică tipul ultimei operații aritmetice efectuate: scădere (N=1) sau adunare (N=0). Acest indicator a fost necesar, deoarece algoritmul de corecție a rezultatului unei operații de adunare a două numere în cod BCD, utilizat de instrucțiunea DAA, este diferit de cel al unei operații de scădere [10].

Indicatorul P/V are dublă funcționalitate. După o operație logică, el indică *paritatea* (Parity) rezultatului: P/V=1 dacă rezultatul are un număr par de biți "1". În cazul unei operații aritmetice, P/V=1 indică *depășirea* (oVerflow) gamei de reprezentare în complement față de 2 a rezultatului, în aritmetica întregilor cu semn pe 8 biți [-128, 127] sau pe 16 biți [-32768, 32767]. Spre exemplu, adunarea numerelor 116 și 12 determină:

$$\begin{array}{r} 116 + \quad 01110100 + \\ 12 \quad = \quad 00001100 \quad , \text{ cu } C=0. \\ \hline 128 \quad = \quad 10000000 \end{array}$$

Dar $10000000_2 = -128_{10}$, deci rezultatul este greșit, fapt semnalizat de indicatorul P/V, care se va poziționa pe "1".

Ca și la microprocesoarele 8080 și 8085, registrele de uz general pot fi manipulate fie ca registre simple (A, B, C, D, E, H, L), fie ca registre pereche, notate la Z80 prin BC, DE, HL. În schimb, registrul dublu format din acumulator și indicatorii de condiție, denumit PSW la procesoarele Intel, este notat la Z80 cu AF (AF').

Blocul de registre conține și un număr de 6 registre speciale, două dintre ele - SP și PC - fiind identice cu cele de la procesoarele 8080/8085. În plus, Z80 conține două registre de 16 biți, IX și IY, denumite *registre index*, care se încarcă cu *adresa de bază* în cazul adresării indexate. Aceste registre măresc considerabil flexibilitatea microprocesorului, mai ales la operarea cu tabele de date. Registrul I (Interrupt Page Address Register), de 8 biți, permite procesorului să apeleze indirect o rutină de tratare a întreruperii, amplasată oriunde în spațiul de 64 Koct de memorie. Acest registru se încarcă cu octetul mai semnificativ al adresei unui tabel ce conține adresele tuturor rutinelor de întrerupere, în timp ce octetul inferior (deplasamentul în tabel) este furnizat de dispozitivul I/E care provoacă întreruperea. Registrul R (memory Refresh register) este folosit pentru reîmprospătarea memoriei RAM dinamice și conține adresa de rând, de 7 biți. După fiecare ciclu fetch, registrul R este incrementat și conținutul său se depune pe liniile A₆+A₀ ale magistralei de adrese, simultan cu activarea unui semnal de reîmprospătare ($\overline{\text{RFSH}}$). Facilitățile menționate

permit proiectantului de sistem să utilizeze memorii RAM dinamice cu un minimum de HW suplimentar. La alte tipuri de procesoare acest lucru nu este posibil decât prin utilizarea unor circuite specializate pentru reîmprospătare (de ex. 8202 sau 8203, produse de Intel).

Ca și 8080/8085, Z80 utilizează două registre "ascunse" de 8 biți, dar care nu sunt accesibile direct prin program, notate W și Z. Aceste registre facilitează efectuarea unor operații interne, servind ca locații temporare la formarea și transferul unor operanzi de 16 biți.

Mecanismul de tratare a întreruperilor utilizează, la Z80, două perechi de bistabile. Prima pereche, IFF1 și IFF2, permite activarea/dezactivarea întreruperilor mascabile, respectiv memorarea lui IFF1 pe durata unei întreruperi nemascabile. Cea de-a doua pereche, IMF_e și IMF_b, face posibilă alegerea prin program a unuia din cele trei moduri de întreruperi mascabile disponibile. Bistabilele IFF sunt setate/resetate prin instrucțiuni EI/DI, identice cu cele de la procesoarele Intel, iar CBB-urile IMF prin instrucțiunile IM 0, IM 1 și IM 2.

5.1.1. Conexiunile externe ale microprocesorului Z80

În fig.5.3 sunt prezentate cele 40 de conexiuni externe ale microprocesorului Z80 [44, 45], a căror funcționalitate este următoarea:

$A_{15} \div A_0$ - magistrala de adrese, cu facilitatea TS, prin intermediul căreia se poate adresa o memorie externă de 64Koct. De asemenea, prin intermediul liniilor $A_7 \div A_0$, microprocesorul poate adresa 256 porturi de intrare și 256 porturi de ieșire. Pe durata unui ciclu de reîmprospătare a memoriilor dinamice, liniile $A_6 \div A_0$ conțin adresa rândului curent.

$D_7 \div D_0$ - magistrala bidirecțională de date, cu facilitate TS, prin intermediul căreia se asigură dialogul cu memoria și cu porturile I/E.

$\overline{\text{CLK}}$ - semnal de tact, de nivel TTL, sub forma unei unde dreptunghiulare simetrice. Acest microprocesor funcționează cu un semnal de sincronizare cu o singură fază, fapt ce a făcut ca, uneori, acesta să fie notat și cu Φ . La primele variante NMOS, perioada semnalului $\overline{\text{CLK}}$ este limitată numai inferior, în funcție de tipul microprocesorului: 400ns la Z80, 250ns la Z80A și 165ns la Z80B. Ca atare, aceste procesoare pot fi comandate și static, stare cu stare, prin comutarea semnalului $\overline{\text{CLK}}$, cu respectarea duratei pali-erului de "0", care nu trebuie să depășească 2 μ s [44].

Ultimele variante NMOS nu mai au această facilitate, în schimb, toate variantele CMOS pot fi comandate static, fără nici o restricție, prin comutarea manuală a semnalului $\overline{\text{CLK}}$ [45].

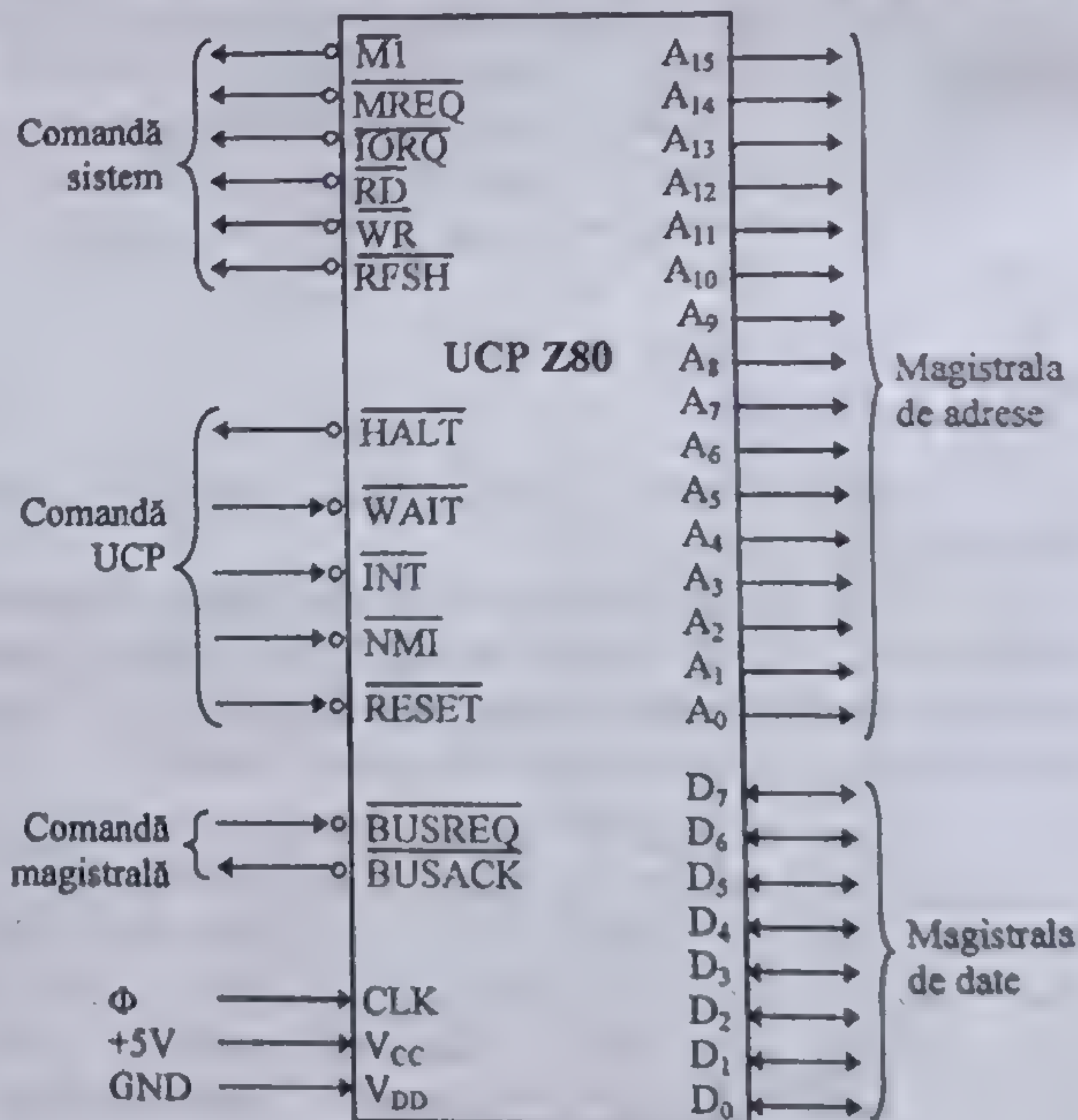


Fig.5.3. Conexiunile externe ale microprocesorului Z80

Partea a II-a - Sisteme cu microprocesoare de 8 biți

Semnale de comandă ale UCP

RESET - este semnalul de inițializare HW a microprocesorului. Pentru o inițializare corectă, acest semnal trebuie să fie activ ("0" logic) minimum $3 \times t_{CLK}$. Pe durata inițializării, magistralele de adrese și date trec în starea de înaltă impedanță, ieșirile de comandă devin inactive și au loc următoarele operații:

- contorul de program, PC, se încarcă cu 0000h;
- se resetează bistabilele IFF1 și IFF2, dezactivându-se întreruperile;
- se șterge conținutul registrelor I și R;
- se setează modul 0 de tratare a întreruperilor ($IMF_a = IMF_b = 0$).

WAIT - semnal activ pe nivel coborât, care permite trecerea microprocesorului în regim de așteptare în cazul în care memoria sau dispozitivele I/E adresate nu sunt pregătite pentru un transfer de date. Pe durata regimului de așteptare, UCP nu realizează refreșarea memoriei dinamice.

NMI (Non Maskable Interrupt) - intrare pentru întreruperi nemascabile, activă pe front descendent. O întrerupere nemascabilă este întotdeauna acceptată la sfârșitul instrucțiunii curente, indiferent de starea bistabilelor IFF, în lipsa unei cereri de cedare de magistrale. Microprocesorul memorează starea întreruperii mascabile (v.fig.5.1b) și forțează un salt la adresa 0066h, unde începe rutina de tratare. Această linie este de prioritate superioară liniei **INT**.

BUSREQ (Bus Request) - intrare pentru comanda trecerii microprocesorului în regimul de cedare de magistrale, echivalentă liniei HOLD de la procesoarele Intel. Cererea este recunoscută la sfârșitul instrucțiunii în curs, are prioritate superioară liniei NMI și asigură trecerea magistrelor de date, de adrese și a liniilor **MREQ**, **IORQ**, **RD** și **WR** în starea de înaltă impedanță. Această intrare este prevăzută cu posibilitatea de implementare a logicii SAU-cablat, fapt ce impune conectarea în exterior a unui rezistor la V_{CC} , atunci când se folosește regimul de cedare de magistrale.

BUSACK (Bus Acknowledge) - este semnalul de confirmare a cedării magistrelor de către UCP, echivalent semnalului HLDA de la procesoarele Intel. Pe durata cedării magistrelor, Z80 nu generează semnale de reîmprospătare a memoriei RAM dinamice.

INT - intrare pentru cereri de întrerupere mascabile, de la dispozitive periferice. Procesorul poate accepta o solicitare pe această linie tot la sfârșitul instrucțiunii curente, dacă bistabilul IFF1 este în prealabil setat și dacă nu există o cerere NMI. Structura internă a microprocesorului asigură și pe această linie o logică de tip SAU-cablat, ceea ce impune în exterior conectarea unui rezistor la V_{CC} .

HALT (Halt State) - ieșire care indică faptul că procesorul a executat o instrucțiune HALT și așteaptă o întrerupere nemascabilă sau mascabilă, dacă mecanismul a fost în prealabil activat. Din starea de oprire, microprocesorul mai poate fi scos și printr-o comandă RESET. Pentru a asigura reîmprospătarea memoriilor dinamice pe întreaga durată a unei stări HALT, Z80 extrage instrucțiuni din memorie, dar va ignora codul de pe magistrala de date, înlocuindu-l intern cu 00h (NOP).

Semnale de comandă ale sistemului

În afară de semnalele strict necesare dialogului cu memoriile SRAM și cu dispozitivele I/E (**MREQ**, **IORQ**, **RD** și **WR**), Z80 mai generează încă două semnale specifice: **M1** și

$\overline{\text{RFSH}}$, care completează funcționalitatea primelor și facilitează utilizarea memoriilor de tip DRAM. Toate cele șase semnale formează *magistrala de comandă* a unui sistem cu Z80.

$\overline{\text{MI}}$ (Machine Cycle One) - ieșire activă pe nivel coborât, care împreună cu $\overline{\text{MREQ}}$ specifică faptul că microprocesorul execută un ciclu mașină de extragere din memorie a opcodului. Dacă apare în conjuncție cu $\overline{\text{IORQ}}$, semnifică începutul unui ciclu de confirmare a acceptării unei întreruperi mascabile.

$\overline{\text{MREQ}}$ (Memory Request) - semnal de ieșire, cu facilitatea TS, care indică faptul că magistrala de adrese conține o adresă de memorie, pentru realizarea unei operații de citire sau scriere.

$\overline{\text{IORQ}}$ (Input/Output Request) - semnal de ieșire, cu facilitatea TS, care indică faptul că liniile $A_7 \div A_0$ conțin adresa unui dispozitiv I/E, în vederea realizării unei operații de citire sau scriere. Dacă este generat împreună cu semnalul $\overline{\text{MI}}$, indică acceptarea de către Z80 a unei întreruperi mascabile și faptul că pe magistrala de date se poate plasa vectorul de întrerupere.

$\overline{\text{RD}}$ (Read) - indică faptul că microprocesorul urmează să efectueze o operație de citire din memoria externă sau dintr-un port I/E. Memoria sau dispozitivul I/E adresat vor utiliza acest semnal pentru a plasa datele pe liniile $D_7 \div D_0$. Are, de asemenea, facilitatea TS.

$\overline{\text{WR}}$ (Write) - ieșire cu facilitatea TS, activată de Z80 pentru a semnala că pe magistrala de date se află un octet ce urmează a fi înscris în locația de memorie sau dispozitivul I/E adresat.

$\overline{\text{RFSH}}$ (Refresh) - ieșire activă pe "0" logic, care împreună cu $\overline{\text{MREQ}}$ indică faptul că pe liniile $A_6 \div A_0$ se află adresa curentă de reîmprospătare a memoriei dinamice, furnizată de registrul R. Linia A_7 este menținută în acest timp în "0", iar pe liniile $A_{15} \div A_8$ se depune conținutul registrului I.

5.1.2. Tratarea întreruperilor la microprocesorul Z80

Deși acest microprocesor are numai două linii externe pentru întreruperi, $\overline{\text{INT}}$ și $\overline{\text{NMI}}$, el posedă un sistem foarte eficient de tratare a cererilor de întrerupere. Or, așa cum s-a mai arătat, caracteristicile de funcționare în timp real a unui sistem cu microprocesor sunt direct influențate de modul în care UCP răspunde la cererile de întrerupere. Cele două tipuri de întreruperi, mascabile și nemascabile, sunt controlate de microprocesor prin intermediul bistabilelor IFF1 și IFF2 (v.fig.5.1b). Aceste fanioane indică starea întreruperilor mascabile, iar modul cum pot fi poziționate (HW sau SW) este arătat în tab.5.1.

Tab.5.1.

Nr. crt.	Acțiunea	IFF1	IFF2	Comentariu
1	Resetare UCP	0	0	Se dezactivează întreruperile mascabile (pe linia $\overline{\text{INT}}$).
2	Execuție instrucțiune DI	0	0	Se dezactivează întreruperile mascabile.
3	Execuție instrucțiune EI	1	1	Se activează întreruperile mascabile.
4	Acceptare întrerupere nemascabilă (pe linia $\overline{\text{NMI}}$)	0	IFF1	Întreruperile mascabile se dezactivează și $\text{IFF1} \rightarrow \text{IFF2}$.
5	Execuție instrucțiune RETN, de revenire din NMI	IFF2	IFF2	$\text{IFF1} \leftarrow \text{IFF2}$.
6	Execuție instrucțiune LD A,I	IFF1	IFF2	$\text{IFF2} \rightarrow \text{P/V}$ (Parity/overflow flag)
7	Execuție instrucțiune LD A,R	IFF1	IFF2	$\text{IFF2} \rightarrow \text{P/V}$

Se observă faptul că o întrerupere nemascabilă blochează acceptarea unei întreruperi mascabile. Prin mecanismul de memorare și refacere a conținutului bistabilului IFF1 (liniile 4 și 5

în tab.5.1), starea sistemului de întreruperi mascabile se conservă. Starea bistabilului IFF2 poate fi testată în urma execuției unei instrucțiuni de transfer în acumulator a registrelor I sau R (vezi liniile 6 și 7 în tab.5.1), când conținutul lui IFF2 este transferat în fanionul de paritate/depășire.

Modul în care microprocesorul Z80 răspunde la cele două tipuri de întrerupere este redat în continuare.

5.1.2.1. Tratarea întreruperilor nemascabile

Întreruperile nemascabile sunt solicitate pe linia $\overline{\text{NMI}}$, activă pe front descrescător și sunt tratate într-un singur mod, descris sintetic prin organigrama din fig.5.4. Dacă nu există o cerere de cedare de magistrală, după terminarea instrucțiunii în curs de execuție, microprocesorul începe execuția unui ciclu mașină special (Non-Maskable Interrupt Request Cycle), în cadrul căruia execută următoarele operații:

- copie în IFF2 conținutul lui IFF1;
- dezactivează întreruperile mascabile (IFF1=0);
- salvează conținutul registrului PC în stivă;
- încarcă contorul de program cu valoarea 0066h;
- rulează rutina de tratare a întreruperii, dispusă la adresa menționată mai sus;
- revine din rutina de întrerupere nemascabilă, cu instrucțiunea RETN (RETurn from Non - maskable interrupt), care are două efecte: reface PC și transferă conținutul lui IFF2 în IFF1.

Rutina de tratare poate fi organizată în același mod ca la procesoarele Intel, dar și mult mai eficient, prin folosirea setului secundar de registre. Este evident faptul că durata rutinei este mult mai mică, fiind reduși la maximum timpii consumați cu salvarea și refacerea registrelor interne.

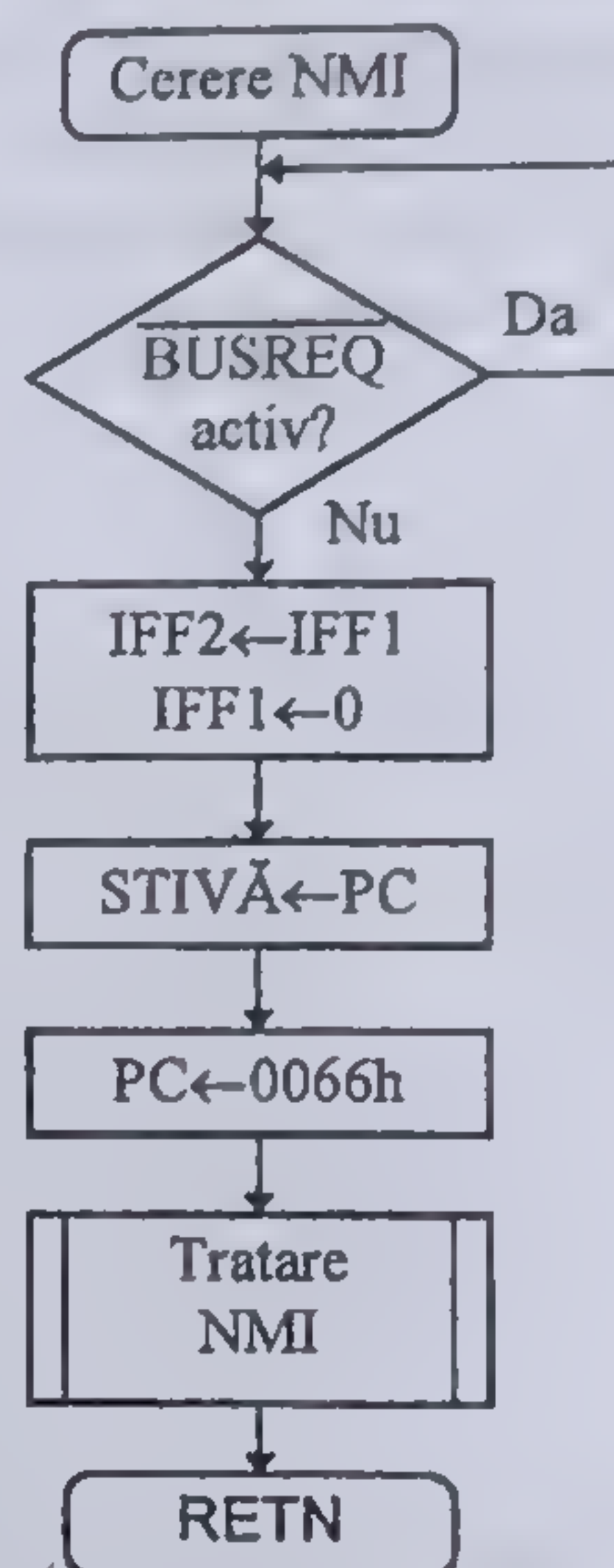


Fig.5.4. Tratarea întreruperilor nemascabile

; UTILIZAREA STIVEI EXTERNE

nmistk:

```

PUSH AF
PUSH BC ; Salvarea registrelor
PUSH DE ; pe stivă.
PUSH HL
; ..... ; Tratare întrerupere
; ..... ; nemascabilă.
POP HL
POP DE ; Refacerea registrelor
POP BC ; de pe stivă.
POP AF
RETN ; Revenire din rutina NMI.
    
```

; UTILIZAREA REGISTRELOR SECUNDARE

nmireg:

```

EX AF,AF' ; Salvarea registrelor
EXX       ; în stiva internă.
; ..... ; Tratare întrerupere
; ..... ; nemascabilă.
EX AF,AF' ; Refacerea registrelor
EXX       ; din stiva internă.
RETN      ; Revenire din rutina
          ; NMI.
    
```

Totuși, nu trebuie abuzat de folosirea repetată, în diferite locuri din program, a instrucțiunilor de schimb între cele două seturi de registre, deoarece nu există nici un indicator al transferurilor interne. Spre exemplu, dacă aceste instrucțiuni se utilizează în rutina de tratare a întreruperii nemascabile, atunci ele nu trebuie să mai apară în altă parte a programului (nici măcar în rutinele de întrerupere mascabile).

5.1.2.2. Tratarea întreruperilor mascabile

Întreruperile mascabile sunt solicitate pe linia \overline{INT} , activă pe nivel coborât. Spre deosebire de cele nemascabile, acestea sunt acceptate numai dacă bistabilul de stare a întreruperilor mascabile este setat ($IFF1=1$). Validarea și invalidarea întreruperilor mascabile se poate realiza prin program, cu ajutorul instrucțiunilor EI, respectiv DI (liniile 2 și 3 în tab.5.1). După resetare, întreruperile mascabile sunt invalidate. Există trei moduri de răspuns, programabile, prezentate în fig.5.5. Modul de întrerupere curent este memorat de bistabilele IMF_a și IMF_b , (v.fig.5.1b).

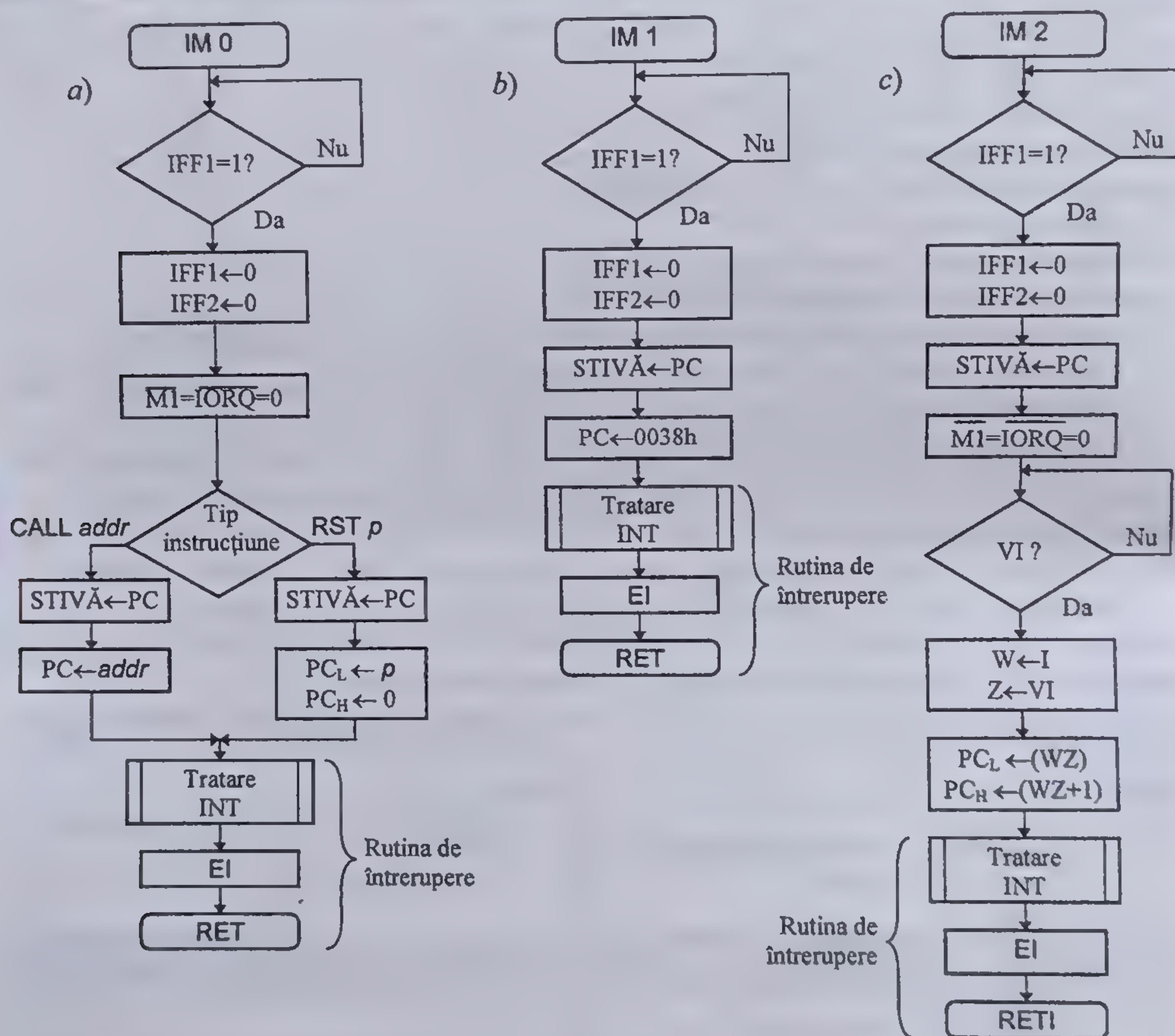


Fig.5.5. Tratarea întreruperilor mascabile în modul 0 (a), modul 1 (b) și modul 2 (c)

a) **Modul 0** - este identic cu modul de răspuns la întreruperi al microprocesorului 8080A. Astfel, Z80 poate trata întreruperi și de la dispozitive I/E din familia Intel. Z80 intră în acest mod după resetare sau prin execuția instrucțiunii cu mnemonica **IM 0** (Interrupt Mode 0). Conform fig.5.5a, la tratarea cererilor în modul 0 microprocesorul Z80 intră într-un ciclu **M1** special, în care realizează următoarele acțiuni:

- dezactivează întreruperile, resetând bistabilele **IFF1** și **IFF2**;
- confirmă acceptarea întreruperii, prin activarea semnalelor $\overline{M1}$ și \overline{IORQ} ;
- acceptă de la dispozitivul solicitant codul instrucțiunii **RST p** (similar cu **RST n** de la 8080/8085, în care $p = n \times 8$) sau al instrucțiunii **CALL addr**, ca răspuns la confirmarea $\overline{M1} = 0$ și

Partea a II-a - Sisteme cu microprocesoare de 8 biți

$\overline{\text{IORQ}}=0$. Deoarece dispozitivele din familia 8080/8085 utilizează o singură linie de confirmare, $\overline{\text{INTA}}$, pentru generarea ei proiectantul de sistem trebuie să introducă o logică suplimentară. Dacă se utilizează instrucțiunea $\text{RST } p$, de 1 octet, pentru preluarea acestuia de către Z80 trebuie generat un singur impuls $\overline{\text{INTA}}$, conform relației: $\overline{\text{INTA}} = \overline{\text{MI}} + \overline{\text{IORQ}}$.

În fig.5.6a, codul instrucțiunii $\text{RST } p$ este furnizat de un port I/E, 8212, funcționând în regimul DATA IN - TS.

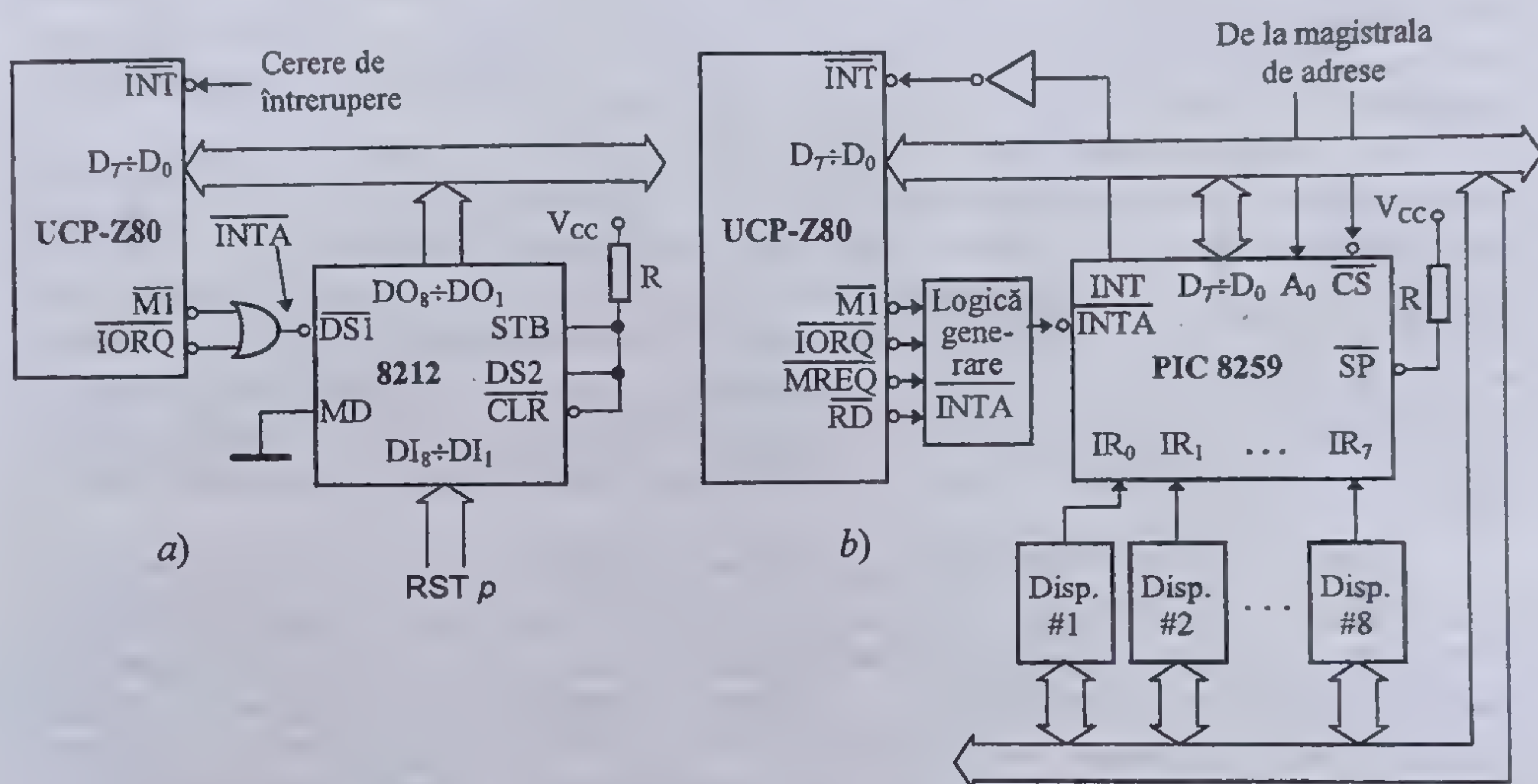


Fig.5.6. Generarea instrucțiunilor $\text{RST } p$ (a) și CALL addr (b), la funcționarea în modul 0

Dacă se utilizează instrucțiunea CALL addr , de 3 octeți, după preluarea codului operație cu primul impuls $\overline{\text{INTA}}$ mai sunt necesare încă două impulsuri, pentru preluarea adresei, ceea ce complică suplimentar logica de generare a semnalului $\overline{\text{INTA}}$. În fig.5.6b, cei trei octeți ai instrucțiunii CALL addr sunt furnizați de un controler de întreruperi 8259, iar semnalul $\overline{\text{INTA}}$ este generat de un automat secvențial simplu, care utilizează semnalele $\overline{\text{MI}}$, $\overline{\text{IORQ}}$, $\overline{\text{MREQ}}$ și $\overline{\text{RD}}$ (vezi și §5.1.3.4).

Apoi, prin execuția instrucțiunii citite în ciclul de confirmare a întreruperii:

- se salvează în stivă contorul de program;
- se încarcă registrul PC cu valoarea p (00h, 08h, ... 38h) sau $addr$;
- se execută rutina de întrerupere și apoi se revine în programul principal. Față de §5.1.2.1, rutina trebuie să conțină și instrucțiunea de reactivare a întreruperilor mascabile, EI, iar în cazul folosirii controlerului 8259 - și comanda de achitare a întreruperii, EOI (v. §4.4.2.1).

b) **Modul 1** - este asemănător răspunsului la o întrerupere NMI, cu diferența că se execută un restart la locația 0038h (fig.5.5b). Intrarea în acest mod se face prin execuția instrucțiunii cu mnemonica $\text{IM } 1$ (Interrupt Mode 1). Faptul că microprocesorul forțează intern adresa rutinei de tratare și nu solicită dispozitivului extern să transmită vectorul de întrerupere, permite conectarea la sistem și a perifericelor care nu fac parte din familia Z80 (non-Z80), ceea ce conduce la creșterea flexibilității sistemelor cu Z80. Răspunsul procesorului este foarte rapid, dar dacă există mai multe surse de întrerupere conectate la linia $\overline{\text{INT}}$, trebuie urmat de o procedură de interogare.

c) **Modul 2** (fig.5.5c) se obține în urma execuției instrucțiunii IM 2 și este specific sistemelor care conțin dispozitive I/E numai din familia Z80. Toate aceste dispozitive se caracterizează prin faptul că dispun de o logică inclusă pe cip care detectează acceptarea unei întreruperi de către procesor ($\overline{M1} = \overline{IORQ} = 0$), arbitrează cererile simultane și răspund cu un vector de întrerupere de 8 biți (VI), preprogramat. La primirea vectorului de întrerupere, care are întotdeauna cel mai puțin semnificativ bit egal cu 0, microprocesorul assemblează în registrele W și Z un pointer de 16 biți (WZ), format din conținutul registrului I (MSB) și VI (LSB). Se obține astfel o adresă, întotdeauna pară, dintr-un tabel ce trebuie să conțină adresele tuturor rutinelor de tratare a întreruperilor. Deoarece VI are de fapt numai 7 biți programabili, rezultă că în modul 2 se pot trata direct un număr de 128 de surse distincte de întrerupere.

O altă particularitate este aceea că logica internă de arbitrare a priorităților permite interconectarea dispozitivelor din familia Z80 într-un lanț de priorități (daisy chain), prioritatea în tratare depinzând de poziția fizică în lanț. Pentru interconectare, aceste dispozitive (CTC-Z80, PIO-Z80, DMAC-Z80, SIO-Z80) dispun de două linii specifice: o intrare - IEI (Interrupt Enable Input) și o ieșire - IEO (Interrupt Enable Output). Primul dispozitiv din lanț este și cel mai prioritar, întrucât are linia IEI conectată permanent la "1" logic. Linia IEO a acestuia se conectează la următorul dispozitiv, de prioritate mai mică ș.a.m.d. În fig.5.7 este considerat un astfel de exemplu, în care dispozitivul CTC are prioritate maximă, urmând apoi, în ordine, SIO, DMAC și PIO.

Toate cele patru dispozitive pot lansa cereri de întrerupere către microprocesor prin intermediul liniilor \overline{INT} , care sunt de tipul cu drenă în gol. Rezistorul R asigură o logică "SAU cablat", prin intermediul căreia orice dispozitiv din lanț poate solicita atenția UCP, spre a fi tratat prin întreruperi. Atât timp cât nici una din componentele lanțului nu cere întrerupere, ieșirile IEO sunt menținute în "1" logic. În momentul în care unul din dispozitive lansează o cerere de întrerupere, își trece linia IEO în "0" logic, fapt care conduce la inhibarea circuitelor din aval, de prioritate inferioară. Z80 răspunde cu $\overline{M1} = 0$, ceea ce nu mai permite modificarea priorităților din lanț. Apoi, trece la execuția unui ciclu FETCH special, de acceptare a întreruperii, cu activarea semnalului \overline{IORQ} în locul semnalului \overline{MREQ} . Până la activarea semnalului \overline{IORQ} , lanțul de priorități IEI-IEO trebuie să se stabilizeze, astfel încât, la îndeplinirea condiției $\overline{M1} = \overline{IORQ} = 0$, numai un singur dispozitiv din lanț va avea IEI=1 și IEO=0. Acesta este dispozitivul solicitant de prioritate maximă, care își va depune pe magistrala de date vectorul de întrerupere. Pentru a asigura timp suficient stabilizării lanțului și plasării vectorului de întrerupere, microprocesorul include automat în ciclul FETCH special două stări de așteptare T_w (v. §5.1.3.4). Același lucru se produce și în modurile 0 și 1 de întreruperi, provocând o creștere a timpului de răspuns cu două cicluri mașină.

Imediat ce dispozitivul apelant a furnizat vectorul de întrerupere, acesta își dezactivează linia \overline{INT} , permițând astfel dispozitivelor mai prioritare din lanț să solicite întreruperi. În schimb, dispozitivele mai puțin prioritare rămân în continuare inhibitate de semnalul IEO=0, care se menține pe toată durata execuției rutinei de tratare a întreruperii. Abia la sfârșitul acesteia, când Z80 extrage din memorie instrucțiunea RETI, dispozitivul cel mai prioritar în curs de servire își re poziționează linia IEO în "1" logic, reconstituind lanțul și permițând circuitelor de mai mică prioritate să fie servite de UCP. Această comportare se datorează faptului că fiecare dispozitiv I/E din familia Z80 conține o logică de decodificare a instrucțiunii RETI, care controlează linia IEO [44, 45]. În acest mod, microprocesorul nu mai trebuie să transmită dispozitivului servit o comandă care să anunțe încheierea tratării, așa cum se face la SINT cu PIC 8259/8259A (EOI), ceea ce mai reduce puțin din timpul de răspuns.

În plus, dispozitivele I/E trebuie să răspundă cu un vector de întrerupere format dintr-un singur octet și nu din doi octeți, ca în cazul sistemelor cu 8080/8085 și PIC, ceea ce constituie un

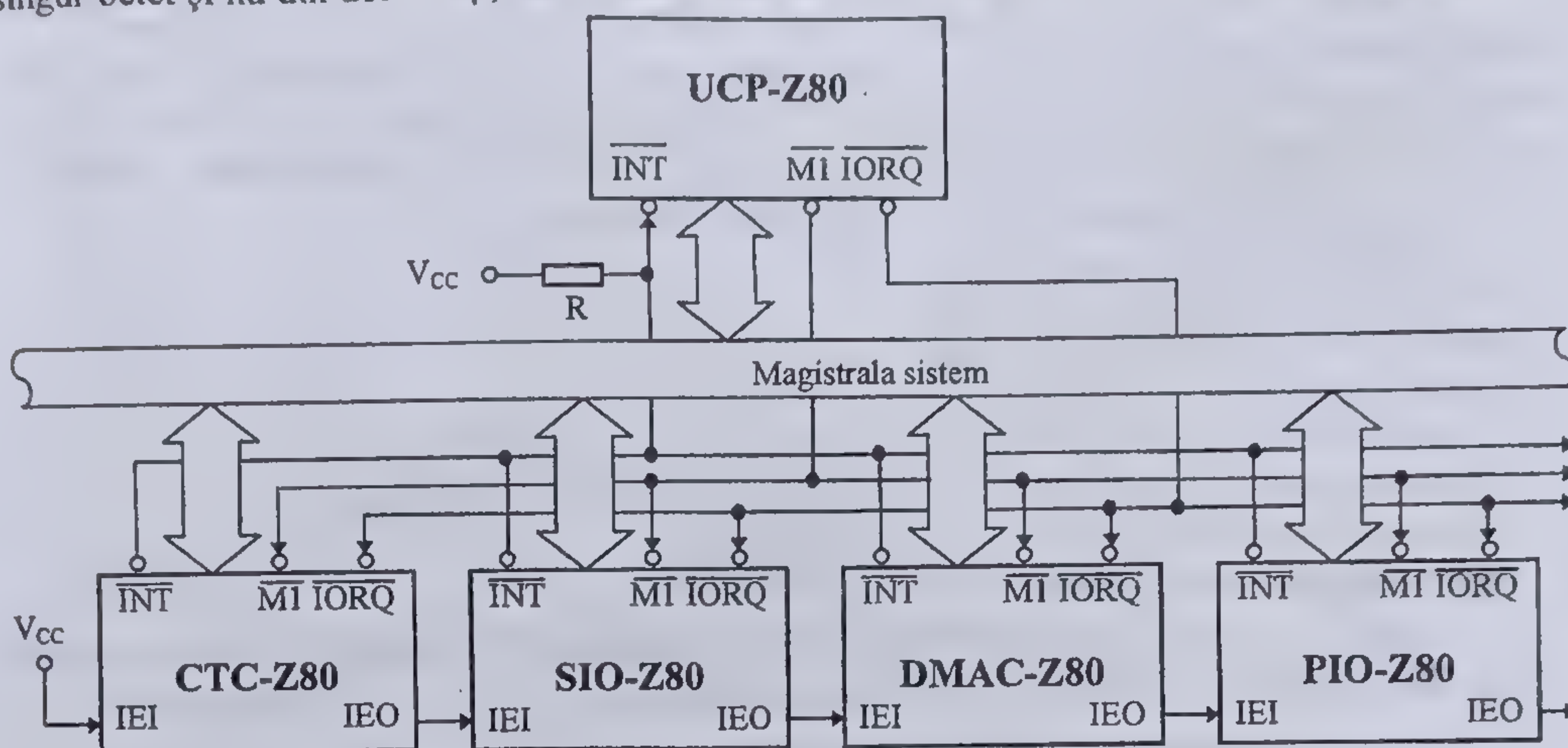


Fig.5.7. Conectarea dispozitivelor din familia Z80 în lanț de priorități specific modului 2 de întreruperi

alt avantaj al acestui mod de întreruperi. Totuși, un dezavantaj îl reprezintă limitarea numărului de dispozitive ce pot fi conectate în lanț, care este determinat de timpul maxim de propagare la stabilirea priorităților. Acest timp poate fi îmbunătățit prin extinderea ciclului de recunoaștere a întreruperii (prin inserarea de stări T_w suplimentare), sau printr-o tehnică de calcul în avans a transportului (carry look ahead).

Spre exemplu, în cazul în care sunt necesare mai mult de patru circuite PIO, pentru stabilizarea lanțului de întreruperi înainte de activarea semnalului \overline{IORQ} , tehnica de calcul în avans a transportului necesită o logică externă ca în fig.5.8.

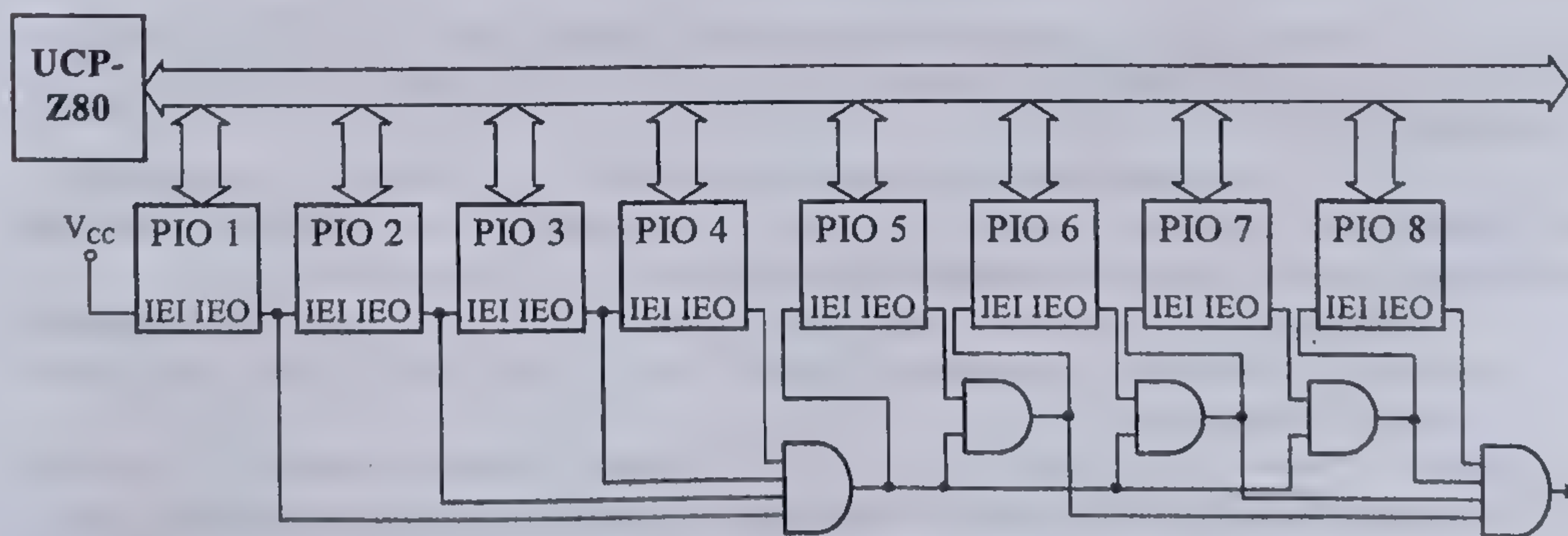


Fig.5.8. Tehnica "carry look ahead", de extindere a lanțului de priorități

Practic, simultan cu transferul semnalelor de validare pentru primele patru circuite, se transmit semnalele și pentru următoarele patru, fără introducerea de întârzieri semnificative. O astfel de procedură face posibilă conectarea până la aproximativ 30 de dispozitive în lanț de priorități [44].

5.1.3. Secvențierea operațiilor interne la microprocesorul Z80

Unitatea elementară de timp în secvențierea operațiilor interne este, ca și la microprocesoarele Intel, perioada unui ciclu de tact (t_{CLK}), numită și stare mașină (T). Un ciclu mașină la acest microprocesor este format din minimum 3 și maximum 6 stări mașină. Pentru execuția celor 158 de tipuri de instrucțiuni, Z80 utilizează nouă tipuri de cicluri mașină, și anume:

- 1) extragere a codului instrucțiunii (Instruction Opcode Fetch),
- 2) citire din memorie (Memory Read)
- 3) scriere în memorie (Memory Write)
- 4) citire dintr-un port de intrare (Input)
- 5) scriere într-un port de ieșire (Output)
- 6) cerere/confirmare întrerupere mascabilă (Interrupt Request/Acknowledge)
- 7) cerere/răspuns la o întrerupere nemascabilă (Non-Mascable Interrupt Request)
- 8) cedare de magistrale (Bus Request/Acknowledge)
- 9) confirmare oprire microprocesor (Halt Acknowledge).

Pentru proiectantul de sistem este necesară cunoașterea particularităților acestor tipuri de cicluri, care îi permite o corectă interconectare a microprocesorului cu dispozitivele externe.

5.1.3.1. Ciclul de extragere a codului operației

Spre deosebire de microprocesoarele Intel de 8 biți, în ciclurile FETCH de la Z80 se realizează două operații: de extragere a opcodului din memorie, respectiv de reîmprospătare a memoriilor RAM dinamice (DRAM). Extragerea codului se efectuează în primele două stări mașină, iar reîmprospătarea în ultimele două. În funcție de starea liniei \overline{WAIT} , pot fi incluse și stări de așteptare:

$$M_1 = \underbrace{T_1 + T_2}_{\text{Extragere opcod}} + \underbrace{T_w}_{\text{Opțional}} + \underbrace{T_3 + T_4}_{\text{Reîmprospătare DRAM}}$$

Trebuie evidențiat faptul că marea majoritate a instrucțiunilor noi ale microprocesorului Z80 au codul operație format din doi octeți și extragerea acestora necesită două cicluri FETCH consecutive.

În fig.5.9 este prezentată cronograma ciclului FETCH, în condițiile în care memoria program impune introducerea unei stări T_w . Pentru o mai bună evidențiere, s-au numerotat principalele faze ale secvențierii operațiilor, și anume:

- 1) În starea T_1 , pe frontul pozitiv al semnalului de tact, este activat semnalul $\overline{M1}$, se depune conținutul numărătorului de program pe liniile $A_{15} \div A_0$, iar liniile magistralei de date trec în starea de înaltă impedanță.
- 2) După aproximativ o jumătate de perioadă, timp necesar stabilizării adresei, Z80 activează semnalele \overline{MREQ} și \overline{RD} , necesare citirii codului instrucțiunii. Aceste semnale, împreună cu $\overline{M1}$, rămân active pe întreaga durată a operației de extragere a opcodului.
- 3) Pe frontul căzător al semnalului de tact din T_2 , microprocesorul testează linia \overline{WAIT} , în vederea sincronizării cu memorii mai lente. Dacă această linie este activată înaintea testării cu cel puțin un interval $t_{SETUP\ WAIT} \geq 70ns$ la Z80 și Z80A, respectiv $\geq 60ns$ la Z80B, microprocesorul introduce stări de așteptare, T_w . Pe frontul descrescător al stărilor T_w se testează din nou linia \overline{WAIT} ; când memoria este pregătită pentru transfer, se trece în starea T_3 .

- 1) Frontul crescător al stării T_3 declanșează citirea codului operației, depus de memorie pe liniile $D_7 \div D_0$, informație care trebuie să fie stabilă cu cel puțin $t_{\text{SETUP DATA}}$ înainte de citire [44, 45]. După extragerea codului instrucțiunii, microprocesorul trece la decodificarea acestuia, își dezactivează semnalele $\overline{\text{MREQ}}$, $\overline{\text{RD}}$, $\overline{\text{M1}}$ și semnalizează trecerea la cea de-a doua parte a ciclului

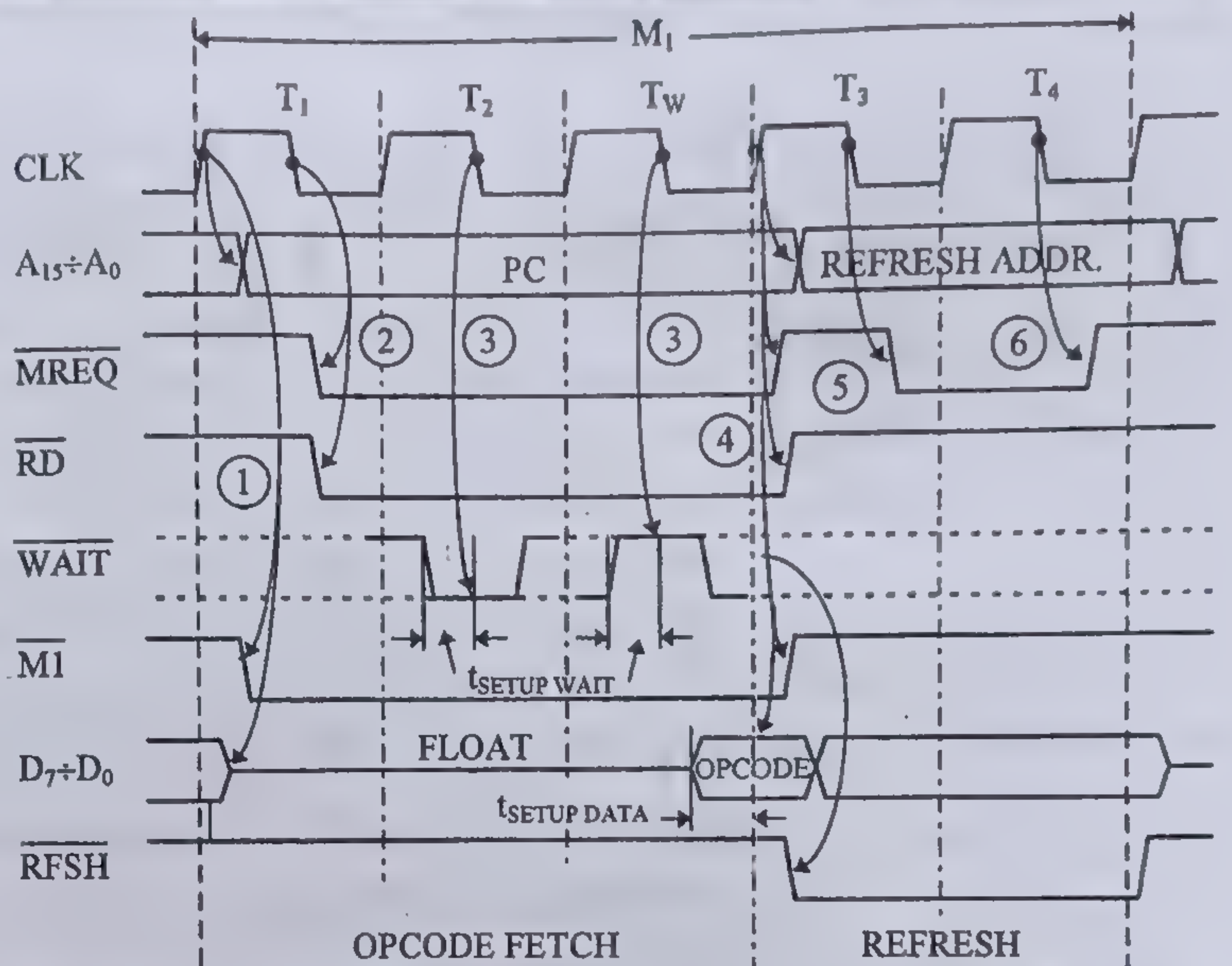


Fig.5.9. Diagrama de timp a ciclului de extragere a codului instrucțiunii

FETCH, activând semnalul $\overline{\text{RFSH}}$. În același timp, Z80 depune pe liniile $A_6 \div A_0$ conținutul registrului R, în vederea reîmprospătării memoriei RAM dinamice.

- 2) Pe frontul descrescător al stării T_3 se reactivează semnalul de dialog cu memoria, $\overline{\text{MREQ}}$, care rămâne activ aproximativ o perioadă de tact.
- 3) În starea T_4 , pe frontul descrescător, se dezactivează semnalul $\overline{\text{MREQ}}$, iar la terminarea stării și semnalul $\overline{\text{RFSH}}$, ceea ce coincide cu sfârșitul operației de reîmprospătare și totodată al ciclului FETCH.

Din analiza secvențierii operațiilor în acest tip de ciclu se pot stabili elementele necesare dialogului UCP cu memoria program (logica de așteptare) și cu memoria de date de tip RAM dinamic.

5.1.3.2. Ciclurile de citire/înscriere a memoriei

Aceste două tipuri de ciclu urmează după ciclul FETCH și comportă trei stări mașină, cu eventuale stări de așteptare, la folosirea memoriilor lente:

$$M_{R/W} = T_1 + T_2 + T_w + T_3$$

opțională

În aceste cicluri nu se execută reîmprospătarea memoriei DRAM, secvențierea operațiilor fiind prezentată în fig.5.10.

Pentru simplitate, s-a considerat o succesiune a celor două tipuri de cicluri, fără stări de așteptare. În ambele situații, microprocesorul depune pe liniile $A_{15} \div A_0$ adresa locației de memorie și apoi, pe frontul căzător din T_1 , activează linia $\overline{\text{MREQ}}$ (faza 1). În același moment, în ciclul de citire activează și semnalul $\overline{\text{RD}}$ (faza 2). În ciclul de înscriere, după activarea semnalului $\overline{\text{MREQ}}$, Z80 depune data ce trebuie înscrisă pe liniile $D_7 \div D_0$ (faza 1). Se observă că în acest ciclu semnalul $\overline{\text{WR}}$ este activat pe frontul căzător al tactului din starea T_2 (faza 2). În ambele tipuri de ciclu,

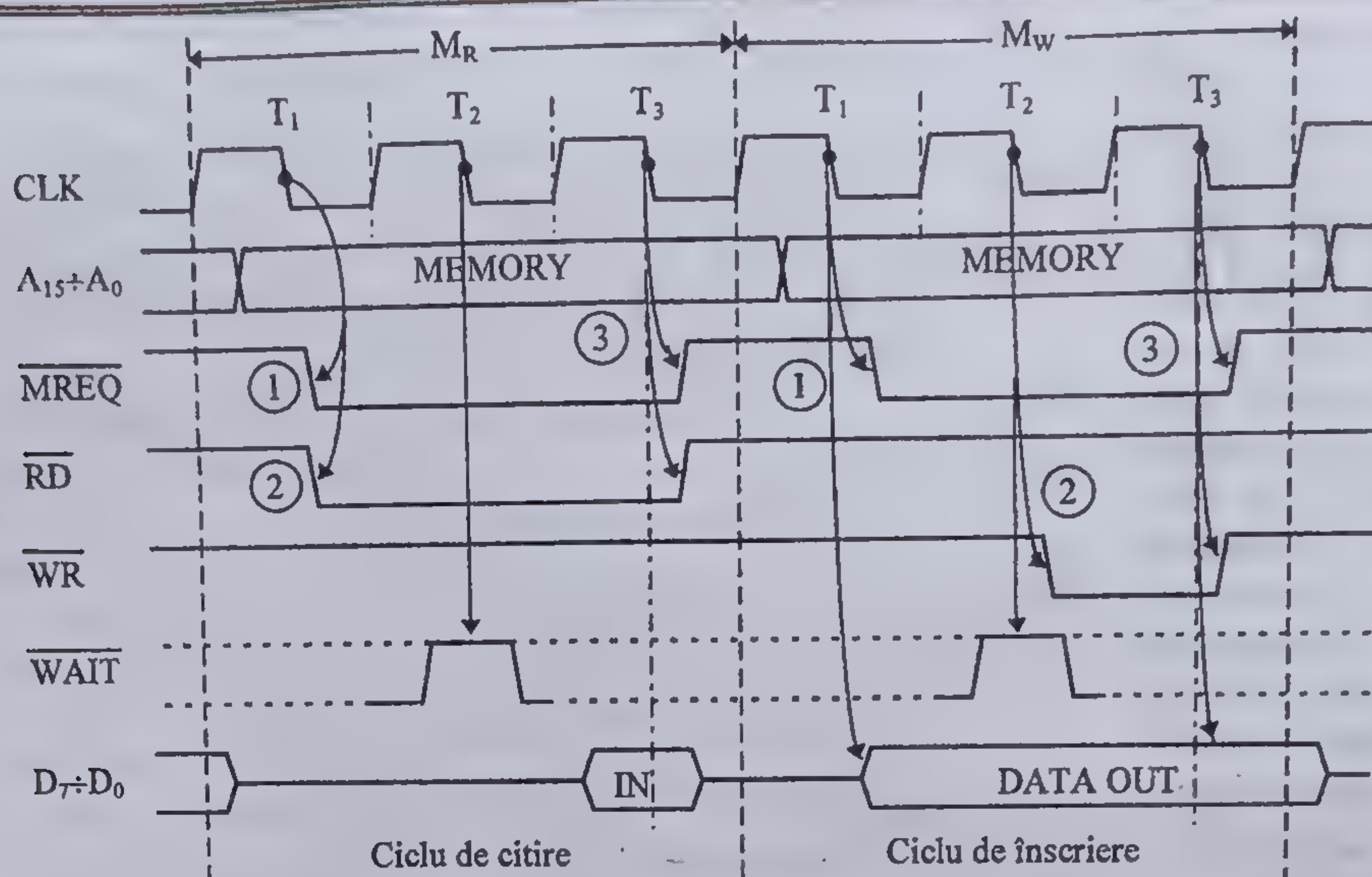


Fig.5.10. Ciclurile de citire/înscriere a memoriei

eșantionarea datelor de pe magistrala de date are loc pe frontul căzător din T_3 , deci cu $1/2T$ mai târziu decât în ciclul FETCH. După citirea/înscrierea datelor, se dezactivează semnalele \overline{MREQ} și $\overline{RD}/\overline{WR}$ (faza 3). Ambele tipuri de ciclu pot fi extinse cu stări T_w , dacă semnalul \overline{WAIT} este găsit activ în T_2 , fiind testat, ca și în ciclul FETCH, pe frontul căzător al semnalului de tact.

5.1.3.3. Ciclurile de intrare/ieșire

În aceste cicluri microprocesorul introduce automat, după T_2 , o stare T_w , care asigură dispozitivelor I/E un timp mai mare cu o perioadă de tact pentru a răspunde unei solicitări a UCP. Dacă este necesar, prin linia \overline{WAIT} se poate solicita introducerea de stări de așteptare suplimentare.

$$M_{I/E} = \underbrace{T_1 + T_2 + T_w}_{\text{opțională}} + \underbrace{T_w + T_3}_{\text{permanente}}$$

Față de ciclurile de citire/înscriere a memoriei,

în acest caz adresa unui port I/E se transmite pe liniile A_7+A_0 . Pe liniile $A_{15}+A_8$ se depune fie conținutul acumulatorului, în cazul instrucțiunilor $IN A, (port)$ și $OUT (port), A$, fie al registrului B,

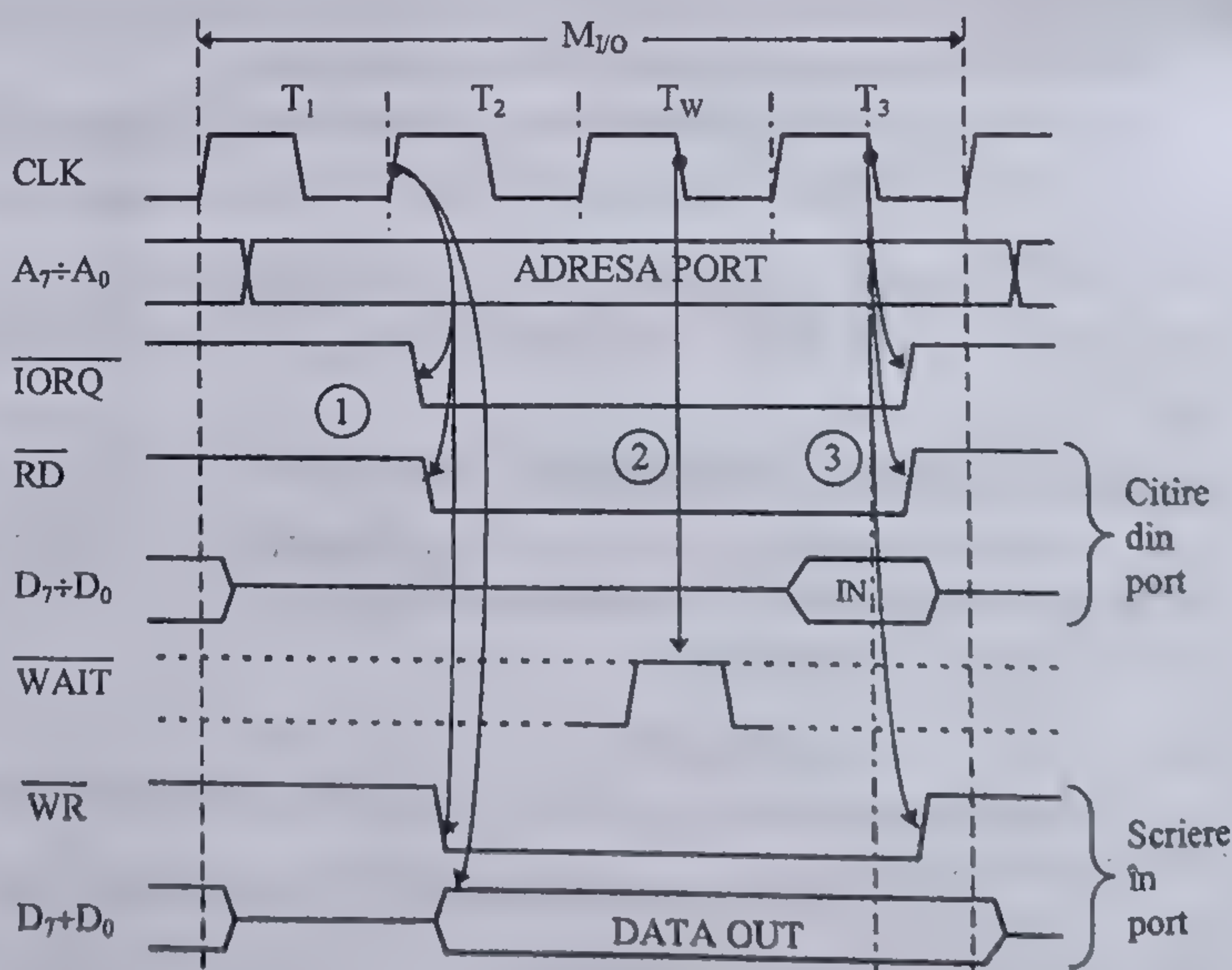


Fig.5.11. Ciclurile de intrare/ieșire (citire/înscriere port)

cazul instrucțiunilor IN r(C), INI, IND, INIR, INDR, OUT (C),r, OUTI, OUTD, OTIR, OTDR (v.tab.5.8).

Secvențierea operațiilor este prezentată în fig.5.11, atât pentru ciclurile de intrare, cât și de ieșire. Trebuie remarcat faptul că, față de ciclurile de dialog cu memoria, activarea semnalelor specifice ($\overline{\text{IORQ}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$) se realizează pe frontul crescător din starea T_2 (faza 1). Linia $\overline{\text{WAIT}}$ este testată pe fronturile descrescătoare ale stărilor T_w , cea obligatorie sau cele opționale (faza 2). Citirea și respectiv înscriserea datelor se fac tot pe frontul căzător din T_3 , ca și în ciclurile de citire/înscrisere a memoriei. După efectuarea acestor operații, semnalele $\overline{\text{IORQ}}$, $\overline{\text{RD}}$ și $\overline{\text{WR}}$ sunt dezactivate (faza 3).

5.1.3.4. Ciclul de cerere/confirmare a întreruperii mascabile

Ca și la microprocesoarele 8080/8085, Z80 testează la sfârșitul fiecărei instrucțiuni linia $\overline{\text{INT}}$, pe frontul ascendent al ultimei stări (faza 1 în fig.5.12). Dacă semnalul $\overline{\text{INT}}$ este activ și dacă mecanismul de întreruperi este validat (IFF1=1), microprocesorul trece la execuția unui ciclu M_1 special, $M_1 \text{ INT}$.

Spre deosebire de ciclurile FETCH, în acest caz microprocesorul introduce automat două stări T_w , după starea T_2 , pentru a permite dispozitivului solicitant să plaseze pe liniile $D_7:D_0$ vectorul de întrerupere (v. §5.1.2). Dacă este necesar, după cele două stări T_w obligatorii pot fi introduse și alte stări T_w , suplimentare:

$$M_1 \text{ INT} = \underbrace{T_1 + T_2 + T_w + T_w}_{\text{obligatorii}} + \underbrace{T_w}_{\text{opțională}} + \underbrace{T_3 + T_4}_{\text{obligatorii}}$$

Secvențierea operațiilor în acest ciclu mașină special decurge în modul următor (fig.5.12):

- se activează semnalul $\overline{\text{MI}}$ și se "îngheață" prioritățile, în vederea stabilizării lanțului (în Modul 2). Acesta trebuie să se

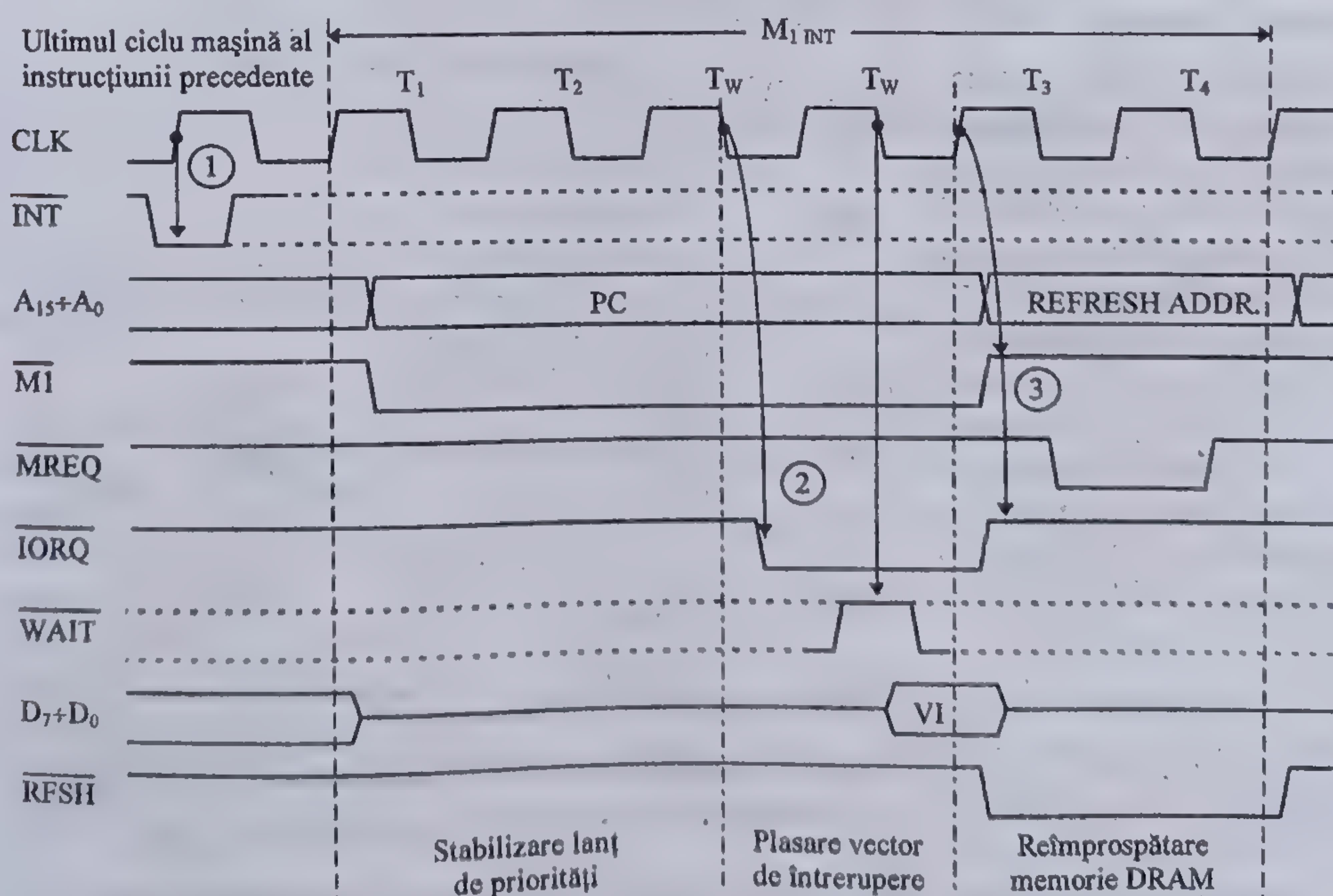


Fig.5.12. Diagrama de semnale pentru ciclul M_1 la acceptarea unei întreruperi mascabile

stabilizeze până la jumătatea primei stări T_W : $t_{STAB} = T_1 + T_2 + \frac{1}{2} T_W$.

- după stabilizarea lanțului de priorități, odată cu frontul căzător al primei stări T_W , se activează semnalul \overline{IORQ} (faza 2), în vederea citirii vectorului de întrerupere (în modul 2), respectiv pentru citirea codului operație al instrucțiunii $RST p$ sau $CALL addr$ (în modul 0).
- primind semnalele $\overline{M1}$ și \overline{IORQ} , de confirmare a acceptării cererii de întrerupere, dispozitivul solicitant plasează pe magistrala de date propriul vector de întrerupere (modul 2), respectiv codul operație al instrucțiunii de apel (modul 0). Dacă dispozitivul I/E nu poate răspunde până la terminarea celei de a doua stări T_W , fapt constatat de microprocesor prin testarea liniei \overline{WAIT} , acesta din urmă introduce un număr corespunzător de stări suplimentare de așteptare.
- pe frontul pozitiv din starea T_3 , microprocesorul citește informația de pe liniile $D_7 \div D_0$, apoi dezactivează semnalele $\overline{M1}$ și \overline{IORQ} (faza 3).
- continuă ciclul $M_1 INT$ cu reîmprospătarea memoriei DRAM, la fel ca în ciclurile M_1 normale (v.fig.5.9).

Reamintim faptul că în modul 1 de întreruperi microprocesorul nu are nevoie de informații de la dispozitivul solicitant și, ca atare, ignoră conținutul magistralei de date.

În modul 2, urmează două cicluri de salvare în stivă a conținutului registrului PC și apoi citirea adresei corespunzătoare din tabelul de întreruperi.

În modul 0, dacă în ciclul $M_1 INT$ s-a citit codul operație al instrucțiunii $CALL addr$, urmează încă două cicluri mașină, de citire a adresei. În aceste cicluri Z80 activează semnalele \overline{MREQ} și \overline{RD} , ca și cum instrucțiunea de apel ar fi extrasă din memorie. În consecință, logica de generare a semnalului \overline{INTA} (v.fig.5.6a) trebuie să inhibe activarea semnalului de citire memorie, \overline{MEMR} și să furnizeze două impulsuri \overline{INTA} suplimentare. Acestea vor fi utilizate de dispozitivul solicitant pentru a plasa pe magistrala de date mai întâi LSB și apoi MSB al adresei rutinei de tratare a întreruperii.

O schemă simplă pentru logica de generare a semnalului \overline{INTA} , împreună cu funcționarea acesteia, sunt prezentate în fig.5.13. Semnalul \overline{MR} este activat în ciclul $M_1 INT$ de conjuncția semnalelor $\overline{M1}$ și

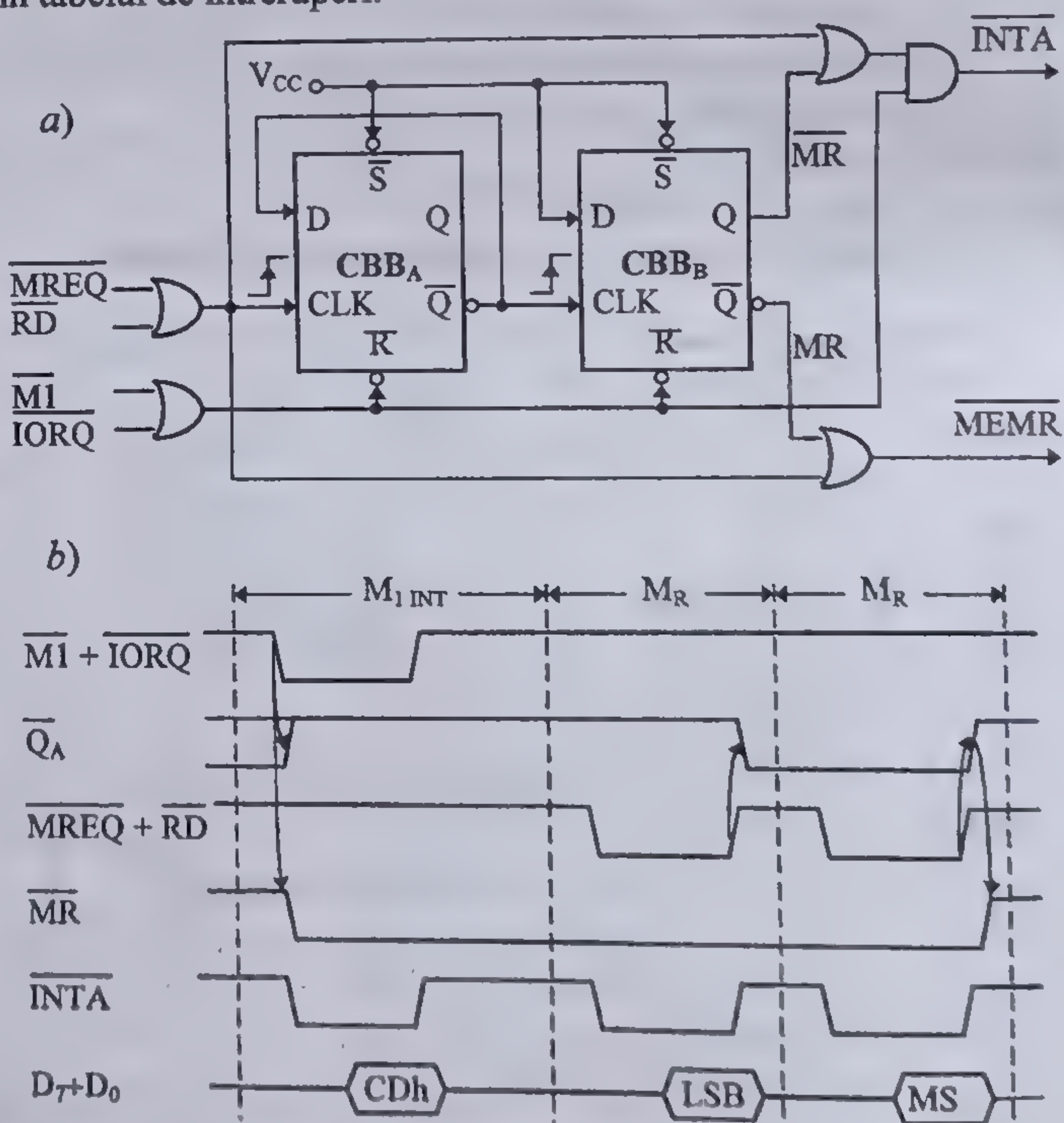


Fig.5.13. Logica de generare a semnalelor \overline{INTA} și \overline{MREQ} în sisteme cu Z80 și PIC8259 (a) și funcționarea acesteia (b)

Fig.5.13. Logica de generare a semnalelor \overline{INTA} și \overline{MREQ} în sisteme cu Z80 și PIC8259 (a) și funcționarea acesteia (b)

$\overline{\text{IORQ}}$ și se menține până la cel de-al doilea front crescător al semnalului compus $\overline{\text{MREQ}} + \overline{\text{RD}}$. Acest semnal validează generarea ultimelor două impulsuri $\overline{\text{INTA}}$, conform relației:

$$\overline{\text{INTA}} = (\overline{\text{MI}} + \overline{\text{IORQ}})(\overline{\text{MREQ}} + \overline{\text{RD}} + \overline{\text{MR}}).$$

În același timp, semnalul MR inhibă activarea liniei $\overline{\text{MEMR}}$, pentru a evita accesul la memoria program:

$$\overline{\text{MEMR}} = \overline{\text{MREQ}} + \overline{\text{RD}} + \text{MR}.$$

Tab.5.2.

Modul de întrerupere	Ciclurile mașină					Comentariu
	M ₁	M ₂	M ₃	M ₄	M ₅	
Modul 0	INTA (6) (SP ← SP-1)	SSH (3) (SP ← SP-1)	SSL (3)	-	-	Dispozitivul I/E furnizează codul instrucțiunii RSTp
	INTA (6)	CML (3)	CMH (4) (SP ← SP-1)	SSH (3) (SP ← SP-1)	SSL (3)	Dispozitivul I/E furnizează instrucțiunea CALL addr
Modul 1	INTA (7) (SP ← SP-1)	SSH (3) (SP ← SP-1)	SSL (3)	-	-	Z80 generează intern instrucțiunea RST 38h
Modul 2	INTA (7) (SP ← SP-1)	SSH (3) (SP ← SP-1)	SSL (3)	CML (3)	CMH (3)	Dispozitivul I/E furnizează, în M ₁ INT, vectorul de întreruperi.

În tab.5.2 se prezintă o sinteză a desfășurării pe cicluri mașină a răspunsului microprocesorului Z80 în cele trei moduri de întrerupere mascabile. Numărul de stări necesare pentru fiecare ciclu este specificat în paranteză.

S-au utilizat următoarele notații:

INTA - ciclu de cerere/confirmare întrerupere (M₁ INT)

SSH - scriere în stivă PC_H (M_w)

SSL - scriere în stivă PC_L (M_w)

CML - citire din memorie, LSB (M_R)

CMH - citire din memorie, MSB (M_R).

5.1.3.5. Ciclul de cerere/răspuns la o întrerupere nemascabilă

Linia $\overline{\text{NMI}}$ este testată de către microprocesor odată cu linia $\overline{\text{INT}}$, la sfârșitul fiecărei instrucțiuni. În plus, starea liniei $\overline{\text{NMI}}$, activă pe front descendent, este memorată într-un bistabil intern. Din acest motiv, semnalul $\overline{\text{NMI}}$ poate avea forma unui impuls de durată t_{NMI} (fig.5.14), nu

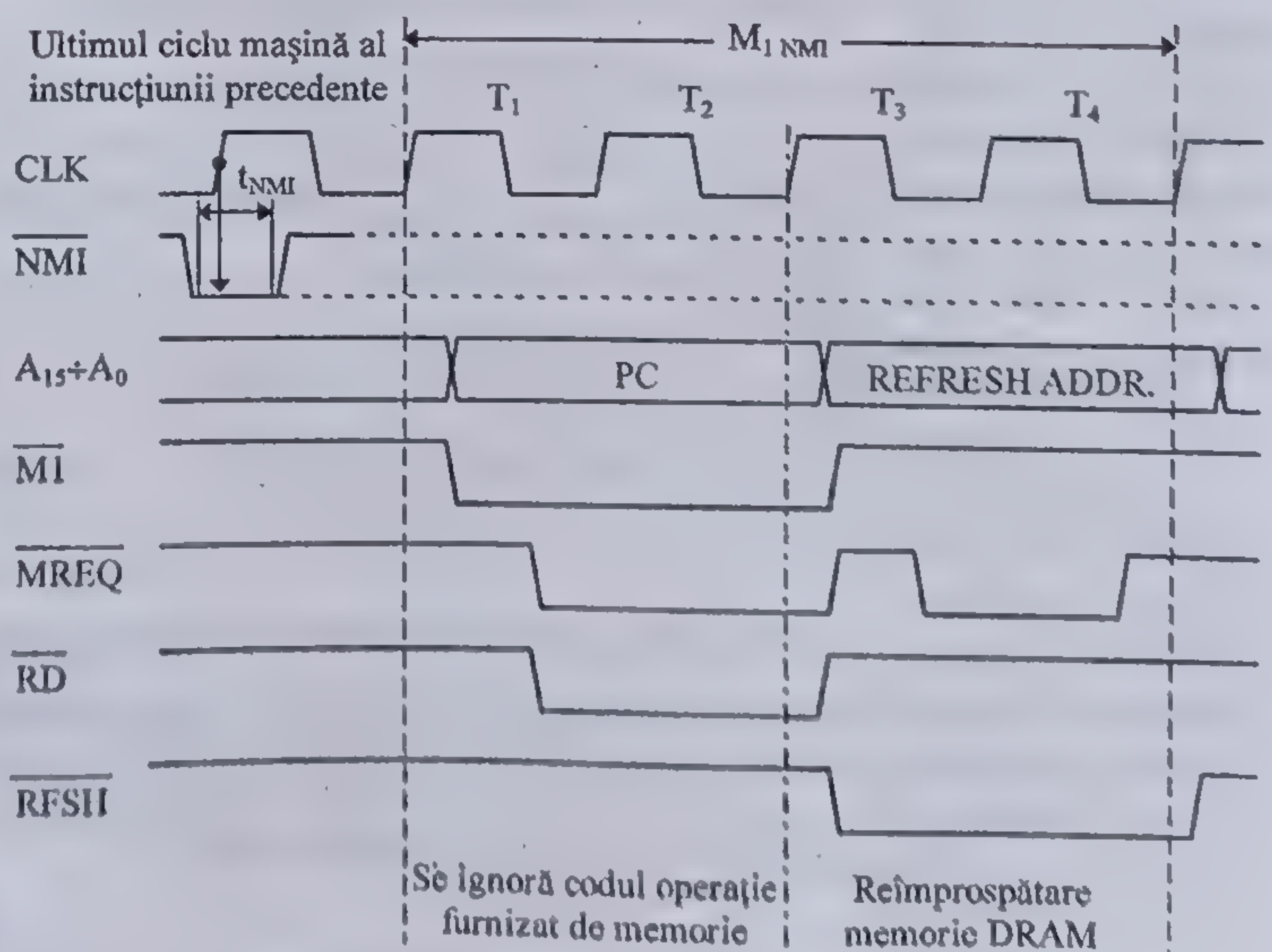


Fig.5.14. Ciclul de cerere/răspuns la o întrerupere nemascabilă

mai mică de 80ns la Z80 și Z80A, respectiv de 70ns la Z80B [44].

Secvențierea operațiilor se aseamănă cu cea de la modul 1 de întrerupere mascabilă, (v.tab.5.2), dar fără stări T_w și fără condiția prealabilă de validare software a mecanismului de întrerupere. De asemenea, trebuie specificat faptul că, în acest caz, nu se confirmă în exterior acceptarea întreruperii.

În fig.5.14 este prezentat primul ciclu mașină, M_{INMI} . El este asemănător unui ciclu FETCH, dar microprocesorul neglijează informația de pe magistrala de date. În schimb, se forțează intern execuția unei instrucțiuni echivalentă cu "restart" (RST), la adresa 0066h (v. §5.1.2).

După operația de reîmprospătare a memoriei dinamice (T_3 și T_4), microprocesorul mai execută încă două cicluri, M_2 și M_3 , pentru salvarea în stivă a conținutului număratorului de program.

5.1.3.6. Ciclul de cerere/confirmare cedare de magistrale

Microprocesorul Z80 testează linia \overline{BUSREQ} tot pe frontul crescător al ultimei stări, însă în fiecare ciclu mașină (fig.5.15). Dacă semnalul este activ, cu respectarea unui interval de prestabilire $t_{SETUP\ BRQ}$ [44], pe frontul pozitiv al următoarei stări mașină, notate T_x , liniile de adresă, de date, precum și liniile de comandă \overline{MREQ} , \overline{IORQ} , \overline{RD} și \overline{WR} sunt trecute în starea de înaltă impedanță (faza 1). Liniile $\overline{M1}$ și \overline{RFSH} , care nu au facilitatea TS, rămân inactive. De asemenea, se activează semnalul de confirmare, \overline{BUSACK} , folosit de noul coordonator pentru preluarea controlului asupra magistralelor sistemului.

Cât timp semnalul \overline{BUSREQ} se menține activ, microprocesorul introduce stări T_x și menține activat semnalul de confirmare, \overline{BUSACK} . Starea liniei \overline{BUSREQ} este testată pe fiecare front pozitiv al semnalului CLK. În momentul în care aceasta a revenit în "1" logic, pe frontul negativ al stării T_x respective este dezactivat și semnalul \overline{BUSACK} (faza 2).

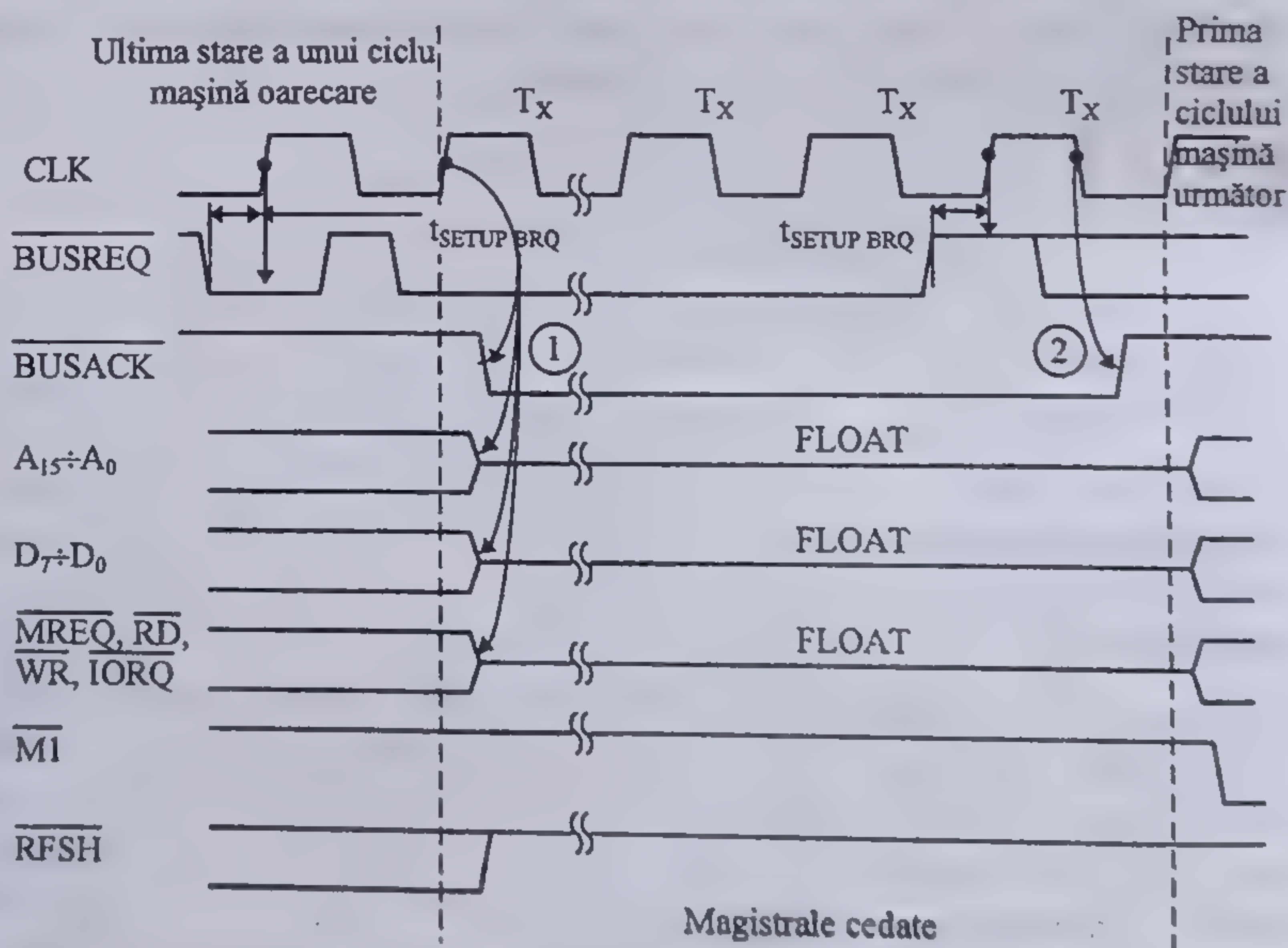


Fig.5.15. Diagrama de semnale în ciclul de cedare de magistrale

În următoarea stare mașină, Z80 preia controlul asupra magistralelor sistemului, continuându-și activitatea cu următorul ciclu mașină, corespunzător punctului de suspendare a activității.

Oprirea UCP pe durata cedării magistralelor impune noului coordonator de sistem să asigure, dacă este necesară, și reîmprospătarea memoriei DRAM.

5.1.3.7. Ciclul de confirmare oprire microprocesor

La extragerea și decodificarea unei instrucțiuni HALT, pe frontul căzător al stării T_4 din ciclul mașină M_1 , microprocesorul Z80 confirmă intrarea într-un ciclu de oprire a procesorului, notat cu $M_{1\text{HALT}}$, în care se activează semnalul $\overline{\text{HALT}}$ (faza 1 în fig.5.16).

Acest ciclu corespunde forțării interne a execuției repetate de instrucțiuni de tip NOP, de 4 stări, pentru asigurarea reîmprospătării memoriilor DRAM din sistem. Ca și la microprocesoarele Intel 8080/8085, ieșirea din starea HALT se poate face prin activarea semnalului $\overline{\text{RESET}}$, ori printr-o cerere de întrerupere, pe liniile $\overline{\text{NMI}}$ sau $\overline{\text{INT}}$. Astfel, se poate realiza o sincronizare a programului cu un eveniment extern sau o blocare a procesorului, în cazul detectării unei erori critice în funcționarea sistemului. Pentru aceasta, în starea T_4 a ciclurilor $M_{1\text{HALT}}$, pe frontul pozitiv al semnalului CLK, sunt testate liniile $\overline{\text{NMI}}$ și $\overline{\text{INT}}$ (faza 2). Dacă unul din aceste sem-

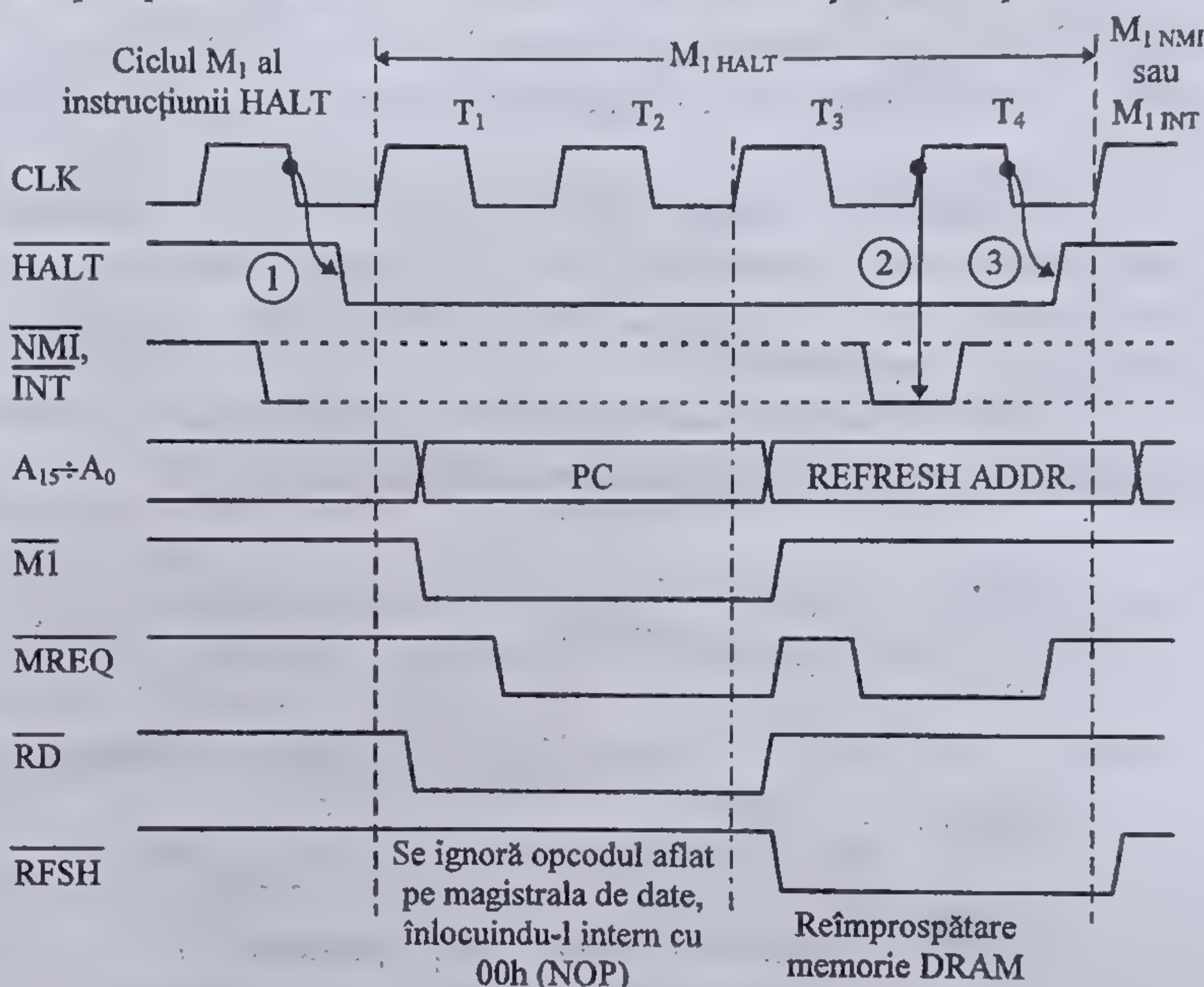


Fig.5.16. Diagrama de semnale în ciclurile de confirmare HALT

nale este activ, pe frontul căzător al stării T_4 din $M_{1\text{HALT}}$ se dezactivează semnalul $\overline{\text{HALT}}$ (faza 3), după care se trece la un ciclu $M_{1\text{NMI}}$, respectiv $M_{1\text{INT}}$.

5.1.4. Conectarea dispozitivelor de memorie la magistralele unui sistem cu Z80

După estimarea necesarului de memorie pentru program și date, trebuie asigurate condițiile unui dialog corect al dispozitivelor de memorie cu UCP (v. §1.1.2.1), prin mixarea semnalului $\overline{\text{MREQ}}$ cu semnalele $\overline{\text{RD}}$ și $\overline{\text{WR}}$, generate distinct la Z80 (v. §5.1.1). În plus, dacă sistemul conține și dispozitive de memorie dinamică, trebuie utilizată logica de reîmprospătare controlată de Z80.

5.1.4.1. Conectarea dispozitivelor de memorie ROM și SRAM

Controlul de către microprocesor al procedurilor de citire/înscrisoare comportă în acest caz două soluții, după cum selecția cipurilor de memorie se face liniar sau cu decodificare.

În cazul *selecției liniare* (sisteme mici sau cipuri de capacitate mare) trebuie asigurate următoarele semnale:

$$\overline{\text{MEMR}} = \overline{\text{MREQ}} + \overline{\text{RD}}, \text{ respectiv}$$

$$\overline{\text{MEMW}} = \overline{\text{MREQ}} + \overline{\text{WR}},$$

ceea ce revine la utilizarea a două porți SAU, ca în fig.5.17.

Liniile de selecție ale dispozitivelor de memorie, $\overline{CS}_1 + \overline{CS}_4$, sunt activate de câte o linie de adresă, în conformitate cu harta memoriei. În cazul memoriilor lente, pentru asigurarea timpului de acces poate fi utilizată o tehnică de așteptare, care comandă linia \overline{WAIT} . În multe situații este necesară introducerea unei stări de așteptare în ciclurile FETCH, unde durata de activare a semnalelor \overline{MREQ} și \overline{RD} este mai mică cu $1/2T$ decât în celelalte cicluri de acces la memorie (v.fig.5.9 și 5.10). Utilizând două bistabile de tip D, sincronizate cu semnalul de tact Φ și interconectate ca în fig.5.18a, se obține pentru semnalul \overline{WAIT} evoluția din fig.5.18b, în fiecare ciclu M_1 .

Semnalul $\overline{M1}$, activat de Z80 la începutul stării T_1 , se transmite la ieșirea Q_1 după frontul crescător din starea T_2 , activând linia \overline{WAIT} . Pe următorul front pozitiv, cel din starea T_w , Q_2 trece la rândul lui în "0" logic, forțând CBB_1 să revină cu $Q_1 = \overline{WAIT}$ în "1" logic, indiferent de durata activării semnalului M_1 .

Dacă sunt necesare stări de așteptare și în ciclurile de citire/înscrisoare a memoriei, poate fi concepută o schemă similară, pilotată de semnalul \overline{MREQ} [10].

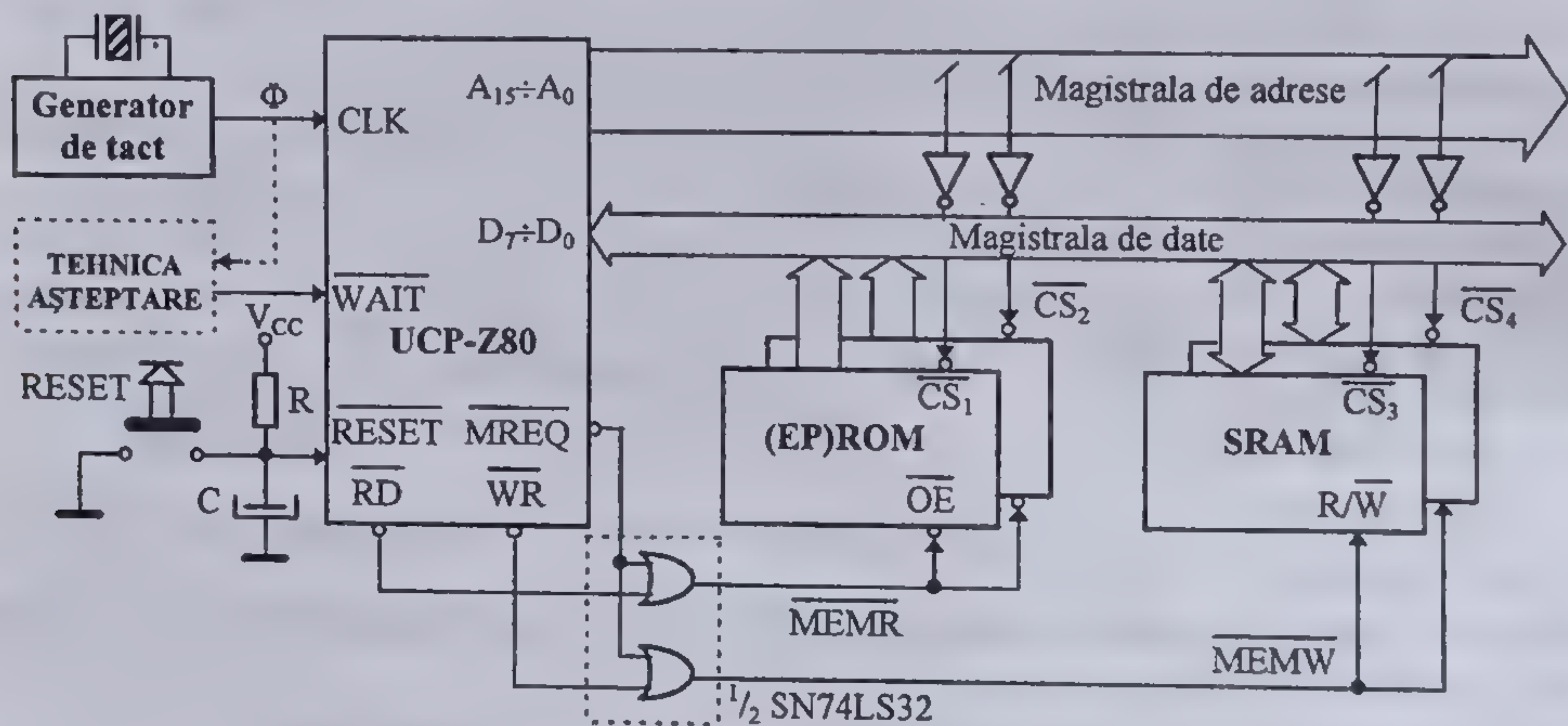


Fig.5.17. Selecția liniară a dispozitivelor de memorie

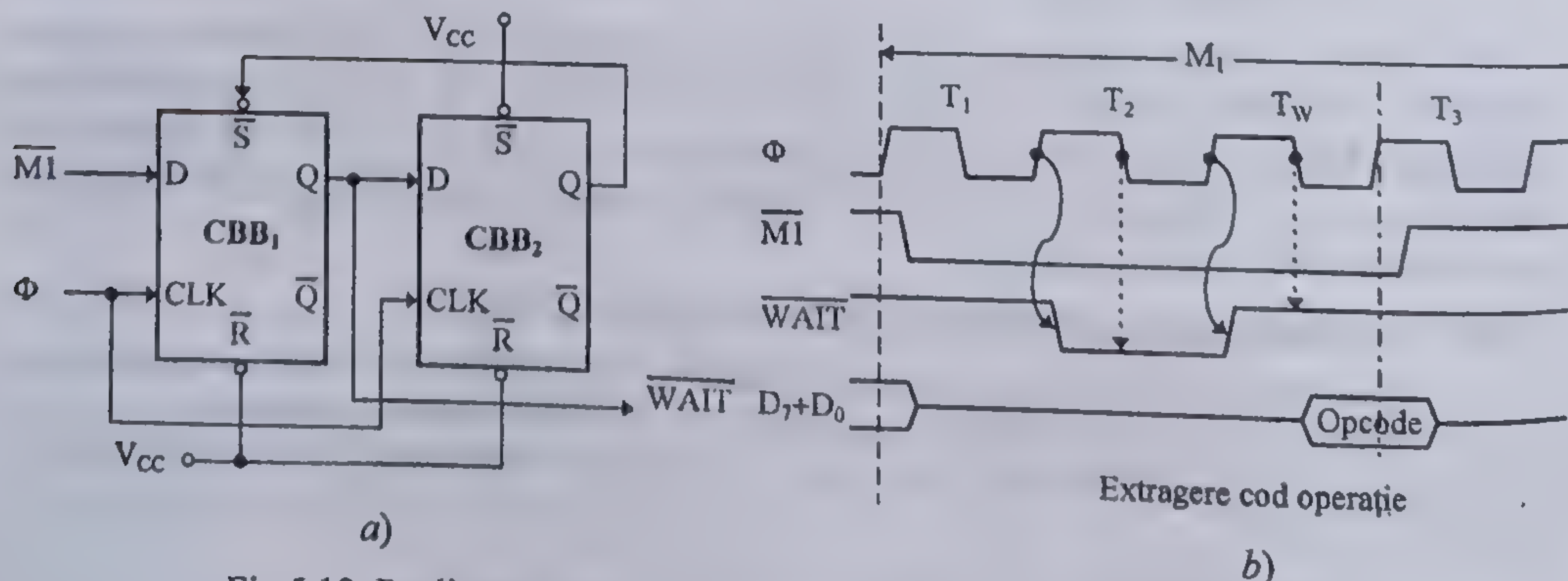


Fig.5.18. Realizarea tehnicii de așteptare pentru ciclurile de tip FETCH

Dacă selecția se realizează prin decodificare, relațiile de mai sus pot fi implementate prin intermediul unui circuit demultiplexor, de ex. 8205 (fig.5.19). Astfel, sunt eliminate porțile necesare în cazul selecției liniare și se poate obține o decodificare completă a adreselor, fără activări redundante a cipurilor de memorie. De asemenea, selecția cu decodificare prezintă și avantajul unei expandări facile a spațiului de memorie.

5.1.4.2. Conectarea memoriilor DRAM

Întrucât Z80 conține pe cip logica de reîmprospătare a memoriilor DRAM, devine avantajoasă implementarea memoriei de date cu astfel de dispozitive. Cu un minimum de circuite auxiliare, se poate conecta la sistem memorie RAM dinamică, chiar până la capacitatea maximă de adresare a microprocesorului, situație întâlnită în multe aplicații.

Exemplu. Se consideră cazul unui sistem cu microprocesor Z80, care necesită o memorie de date (MD) de 64 Kocteți.

Cunoscând principiile de funcționare ale memoriilor RAM dinamice (v. §1.1.2.3), MD se va realiza cu circuite DRAM de tip 2164, cu capacitatea de 64 Kbiți fiecare (fig.5.20).

Deși microprocesorul Z80 dispune de un spațiu logic de adresare de cel mult 64Kocteți, vom considera că sistemul conține mai mult de 64Kocteți de memorie fizică. Aceasta deoarece trebuie să avem în vedere faptul că este absolut necesară existența unui program rezident într-o memorie de tip ROM, care să preia controlul după resetare.

Logica de decodificare și de generare a semnalelor de selecție pentru memoria fizică nu este în întregime cablată, ci are și o porțiune programabilă, cu rol de configurare a spațiului de memorie fizică ce poate fi "văzut" de către microprocesor la un moment dat. În cele ce urmează vom considera că \overline{CS}_{DRAM} este linia activată de către această logică de configurare, în momentele în care microprocesorul accesează blocul de 64Kocteți de memorie DRAM.

Pentru implementarea fizică a acestui bloc sunt necesare 8 circuite DRAM 2164, care vor fi citite și înscrise simultan, pentru a se obține un cuvânt de date cu lungimea de 8 biți. Pentru multiplexarea celor 16 linii de adresă se vor folosi 2 multiplexoare de tip SN74157, care conțin fiecare câte 4 multiplexoare cu două căi. Ele vor furniza memoriei DRAM adresa de rând (A_7+A_0), când linia de selecție comună $S=0$, respectiv adresa de coloană ($A_{15}+A_8$), când $S=1$.

Semnalul \overline{RAS} trebuie să devină activ odată cu semnalul \overline{MREQ} , dacă în prealabil a fost activat unul din semnalele \overline{CS}_{DRAM} sau \overline{RFSH} .

În lipsa unui acces la memorie (citire, scriere sau reîmprospătare), bistabilul din fig.5.20 este comandat numai pe intrarea sincronă de tip D ($\overline{RFSH} = \overline{R} = 1$) și are $Q=0$, deoarece $\overline{MREQ}=1$. Aceasta face ca linia $S=0$ a multiplexorului să aducă adresa de rând, A_7+A_0 , la intrările de adresă ale memoriei DRAM.

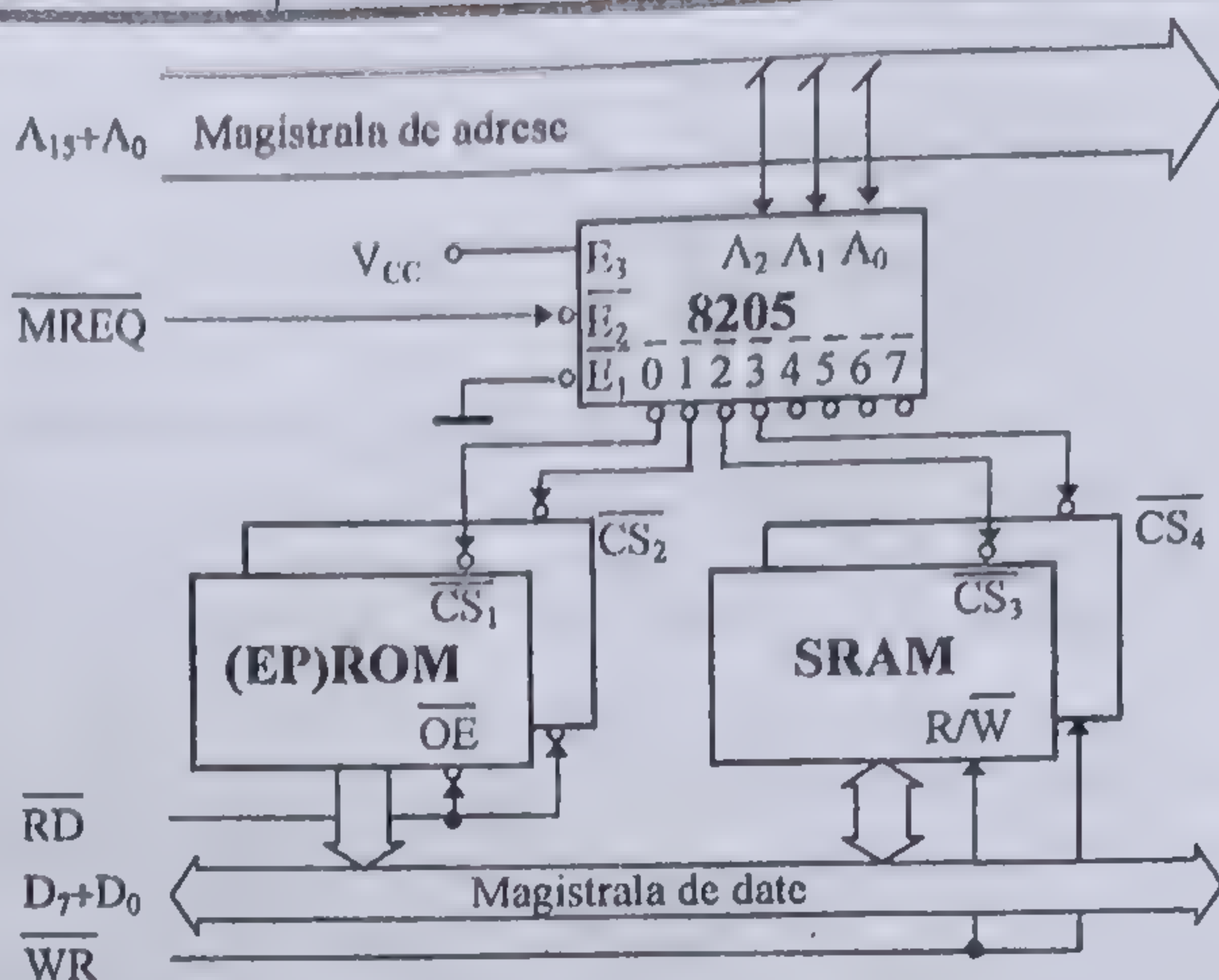


Fig.5.19. Selecția cu decodificare a dispozitivelor de memorie

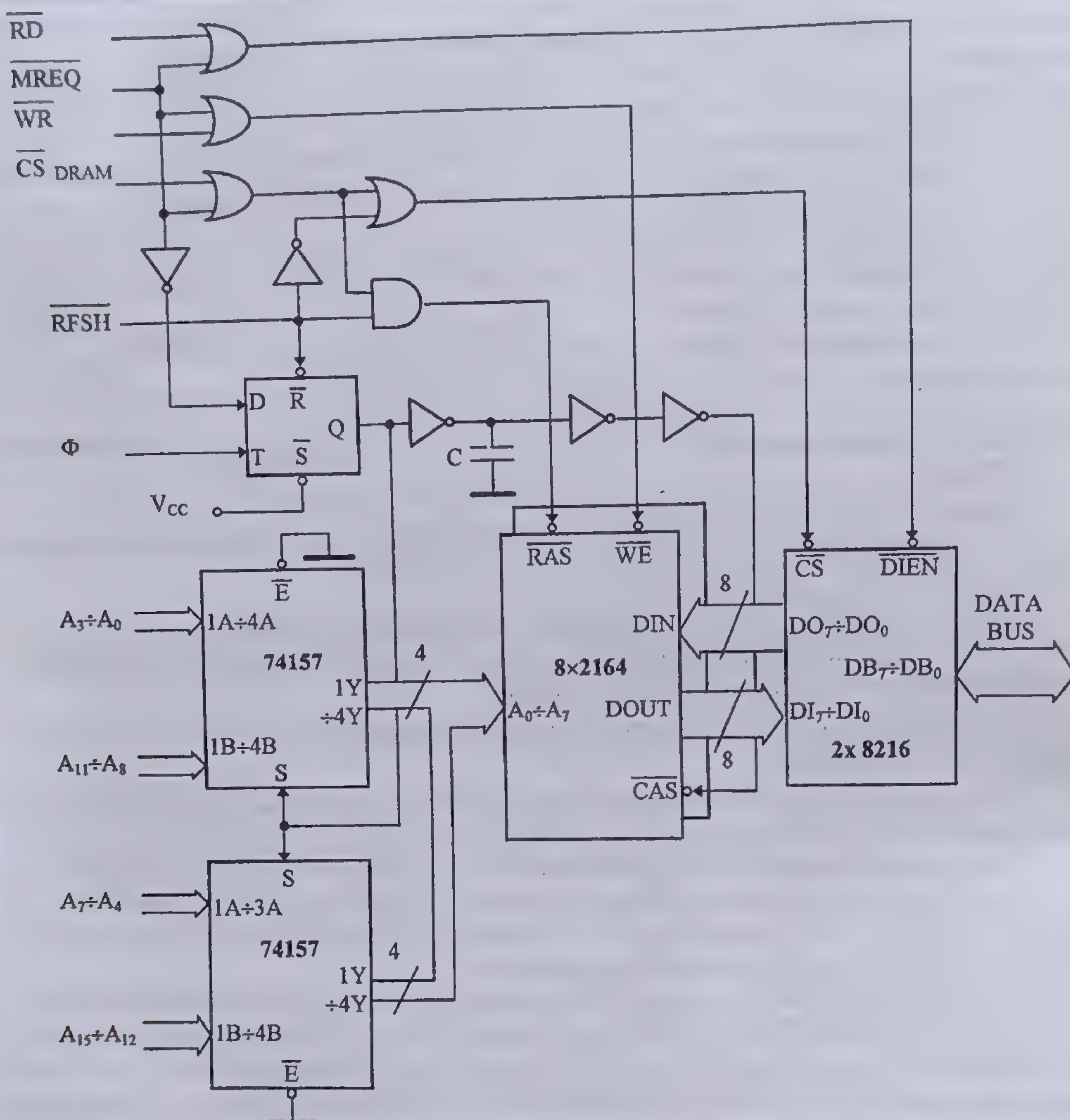


Fig.5.20. Conectarea unui bloc de 64 Kocteți DRAM într-un microsistem cu Z80

La începutul unui acces de citire/scriere din/în blocul de 64Ko de DRAM, $\overline{CS_DRAM} = 0$, $\overline{MREQ} = 0$ și $\overline{RFSH} = 1$. Aceste semnale validează transferul informației între magistrala de date și memoria DRAM prin cele două circuite 8216 ($\overline{CS} = 0$), în sensul stabilit de linia \overline{DIEN} ($\overline{DIEN} = 0$ pentru citire, $\overline{DIEN} = 1$ pentru scriere). În același moment, este activat și semnalul de selecție rând, \overline{RAS} , adresa de rând fiind deja prezentă la intrările de adresă, $A_7 \div A_0$, ale memoriei DRAM. Pe primul front crescător al semnalului de tact, Φ , de după activarea liniei \overline{MREQ} , starea bistabilului este actualizată ($Q = S = 1$). După o întârziere dată de timpul de comutare, multiplexorul aduce adresa de coloană, $A_{15} \div A_8$, la intrările de adresă ale memoriei DRAM. După un interval de timp Δt de la comutarea bistabilului, dat de un circuit de întârziere realizat cu trei porți inversoare și condensatorul C , când adresa de coloană a ajuns deja la memorie, este activat și semnalul \overline{CAS} , selecția celei de memorie DRAM fiind acum completă. Tipul operației efectuate (citire sau

scriere) este selectat la nivelul memoriei DRAM pe linia \overline{WE} , iar data este transferată prin circuitele 8216, validate încă de la începutul ciclului de citire/scriere. Apoi, linia \overline{MREQ} este dezactivată și, la următorul front crescător al semnalului de tact Φ , bistabilul revine în starea inițială ($Q=0$). Multiplexorul reselectează liniile inferioare, $A_7 \div A_0$, pregătindu-se pentru un nou acces la memorie.

Logica de configurare nu intervine în ciclul de reîmprospătare ($\overline{CS}_{DRAM}=1$), deoarece nu necesită un transfer de date pe magistrală. La începutul unui ciclu de reîmprospătare este depusă pe liniile $A_6 \div A_0$ adresa de refreșare (conținutul număratorului de 7 biți reprezentat de registrul intern R al microprocesorului), apoi sunt activate semnalele \overline{MREQ} și \overline{RFSH} . $\overline{RFSH}=0$ determină blocarea oricărui transfer de date prin cele două circuite 8216 ($\overline{CS}=1$), ceea ce face ca ieșirile $DB_7 \div DB_0$ să rămână în starea de înaltă impedanță (HZ) pe toată durata ciclului de refreșare. De asemenea, $\overline{RFSH}=0$ determină activarea semnalului \overline{RAS} , când adresa de refreșare este deja prezentă la intrările de adresă $A_6 \div A_0$ ale memoriei DRAM. În același timp, $\overline{RFSH}=\overline{R}=0$ blochează actualizarea bistabilului pe linia sincronă de date, D. Drept urmare, $Q=S=0$ se va păstra până la terminarea ciclului de refreșare (când $\overline{RFSH}=\overline{MREQ}=1$), inhibând astfel activarea semnalului \overline{CAS} .

Obs.: Memoriile DRAM 2164, de 64Kbiți, sunt organizate intern sub forma a patru bancuri de memorie (4 matrice 128×128). Rândurile de același rang sunt selectate simultan în toate cele 4 bancuri, ceea ce face posibilă refreșarea cu o adresă de doar 7 biți, $A_6 \div A_0$. Bancul de memorie este selectat de cei mai semnificativi doi biți de adresă, A_{14} și A_{15} , prin decodificare. Restul biților de adresă, $A_{13} \div A_7$, vor selecta, după o decodificare prealabilă, una din cele 128 de coloane ale bancului selectat cu biții A_{14} și A_{15} . Se observă, deci, că bitul de adresă A_7 , deși contribuie la selecția coloanei, este reținut în memoria DRAM odată cu adresa de rând, la activarea semnalului \overline{RAS} . Utilizarea lui va fi amânată însă, până ce și restul biților adresei de coloană vor fi reținuți în memoria DRAM, odată cu activarea semnalului \overline{CAS} , tot atunci când sunt memorați și biții de selecție a bancului de memorie, A_{14} și A_{15} .

Toate aceste detalii de funcționare sunt "ascunse" în interiorul cipului de memorie DRAM, astfel încât, din exterior, putem considera că acesta este organizat sub forma unei singure matrice cu 256 de linii și 256 de coloane, ca și cum atât adresa de rând, cât și cea de coloană, ar avea 8 biți. Ele au fost totuși prezentate pentru a justifica lungimea adresei de refreșare (adresa de rând) de numai 7 biți și compatibilitatea mecanismului de refreșare al microprocesorului Z80 cu memoriile DRAM care conțin un număr mai mare de 128×128 de celule, cum este și cazul circuitelor 2164.

5.1.5. Conectarea dispozitivelor I/E

Referitor la conectarea porturilor I/E la magistralele unui sistem cu microprocesor Z80, se pot distinge două situații, după cum acestea fac sau nu parte din familia Z80.

5.1.5.1. Conectarea dispozitivelor I/E non-Z80

Ca și în cazul dispozitivelor de memorie, la utilizarea dispozitivelor I/E care nu fac parte din familia Z80 (cel mai adesea din familia Intel) trebuie asigurate semnalele corespunzătoare de control. Obişnuit, aceste dispozitive necesită semnale unice pentru citire (\overline{IOR}) și pentru înscrisere (\overline{IOW}). În concordanță cu secvențierea operațiilor din ciclurile de intrare-ieșire ale microprocesorului Z80 (v. §5.1.3.3), se pot scrie următoarele relații:

$$\overline{\text{IORD}} = \overline{\text{IORQ}} + \overline{\text{RD}}, \text{ respectiv}$$

$$\overline{\text{IOWR}} = \overline{\text{IORQ}} + \overline{\text{WR}}.$$

Pentru implementare, se pot folosi două modalități de conectare a dispozitivelor I/E non-Z80 la liniile de comandă, în funcție de tipul selecției: liniară sau cu decodificare, așa cum se arată în fig.5.21.

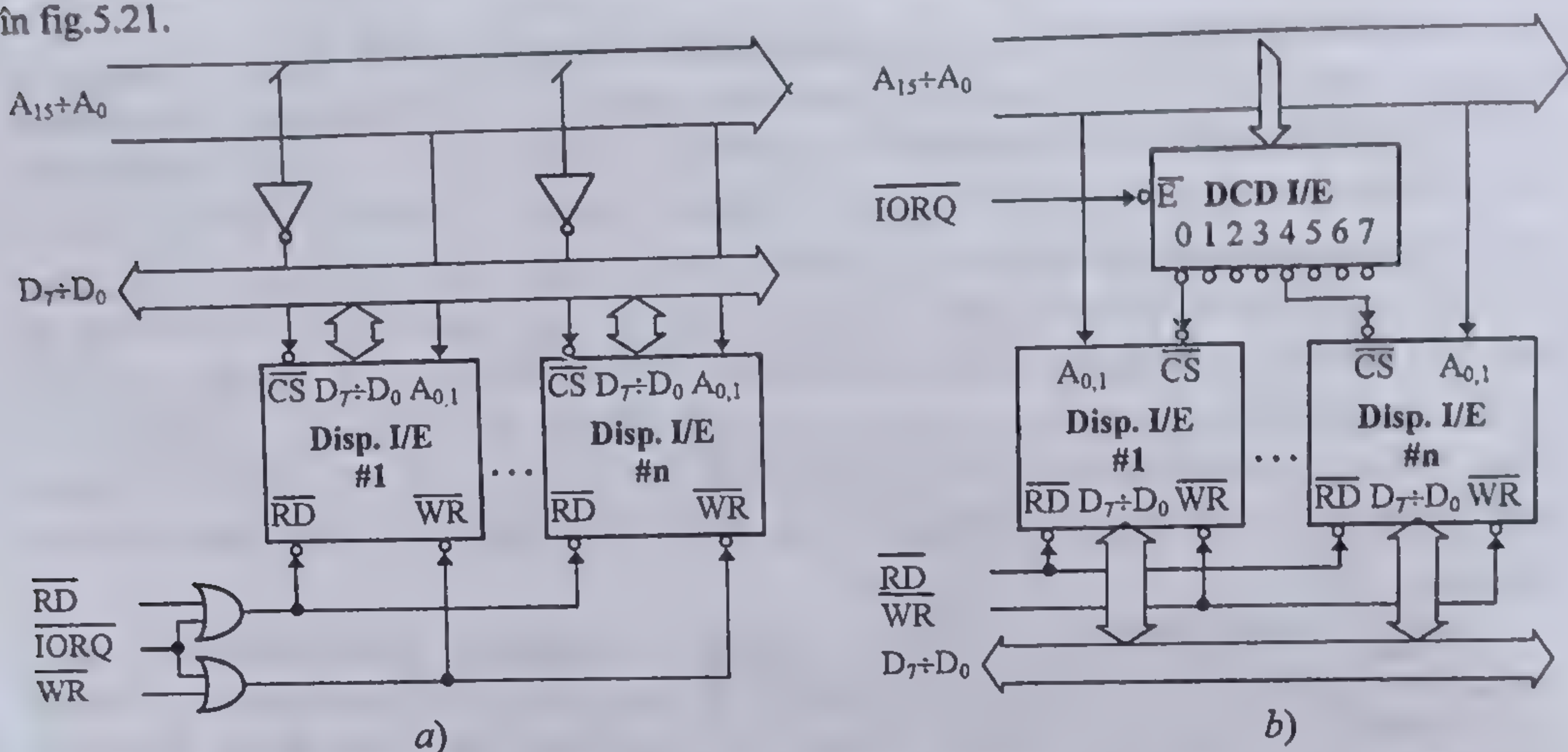


Fig.5.21. Modalități de conectare a dispozitivelor I/E care nu fac parte din familia Z80

Pentru adresare sunt utilizate numai liniile $A_7 \div A_0$, iar pentru selecția registrelor interne ale dispozitivelor se folosesc una sau două din liniile inferioare ale magistralei de adrese: A_0 și/sau A_1 .

5.1.5.2. Conectarea dispozitivelor I/E din familia Z80

Aceste dispozitive sunt prevăzute cu o logică internă de citire/înscrisoare similară cu cea a microprocesorului, care permite conectarea lor direct la magistralele sistemului, fără a fi necesare circuite suplimentare pentru mixarea semnalelor de comandă.

După cum s-a văzut și în §5.1.2.2 (fig.5.7), dispozitivele I/E din familia Z80 posedă liniile $\overline{\text{M1}}$ și $\overline{\text{IORQ}}$, dar și $\overline{\text{RD}}$ (majoritatea), respectiv $\overline{\text{WR}}$ (de obicei, acesta din urmă se generează intern, din $\overline{\text{RD}}$, cu excepția DMAC-Z80). Selecția poate fi liniară sau, cel mai adesea, cu decodificare, așa cum se arată în fig.5.22.

În corespondență cu funcția circuitelor utilizate și cu aplicația dată, în afara conectării la UCP, proiectantul trebuie să asigure interconectarea lor. În exemplul considerat, circuitul CTC-Z80 este utilizat atât pentru contorizarea/măsurarea evenimentelor/intervalelor de timp din sistem (CH#0, CH#1), cât și pentru pilotarea canalelor seriale ale circuitului SIO-Z80 (CH#2, CH#3). Acesta din urmă poate controla un transfer serial cu exteriorul (prin canalul A) sau accesul direct la memorie (canalul B), prin intermediul unui controler DMA. Simplitatea interconectării este remarcabilă, ca și în cazul folosirii modului 2 de întrerupere.

Proiectarea sistemelor cu astfel de dispozitive se reduce la alegerea celui mai avantajos regim de funcționare pentru aplicația dată și la programarea lor corespunzătoare. Pentru aceasta este necesară o cunoaștere detaliată a dispozitivelor din familia Z80, ceea ce se va urmări în capitolele următoare.

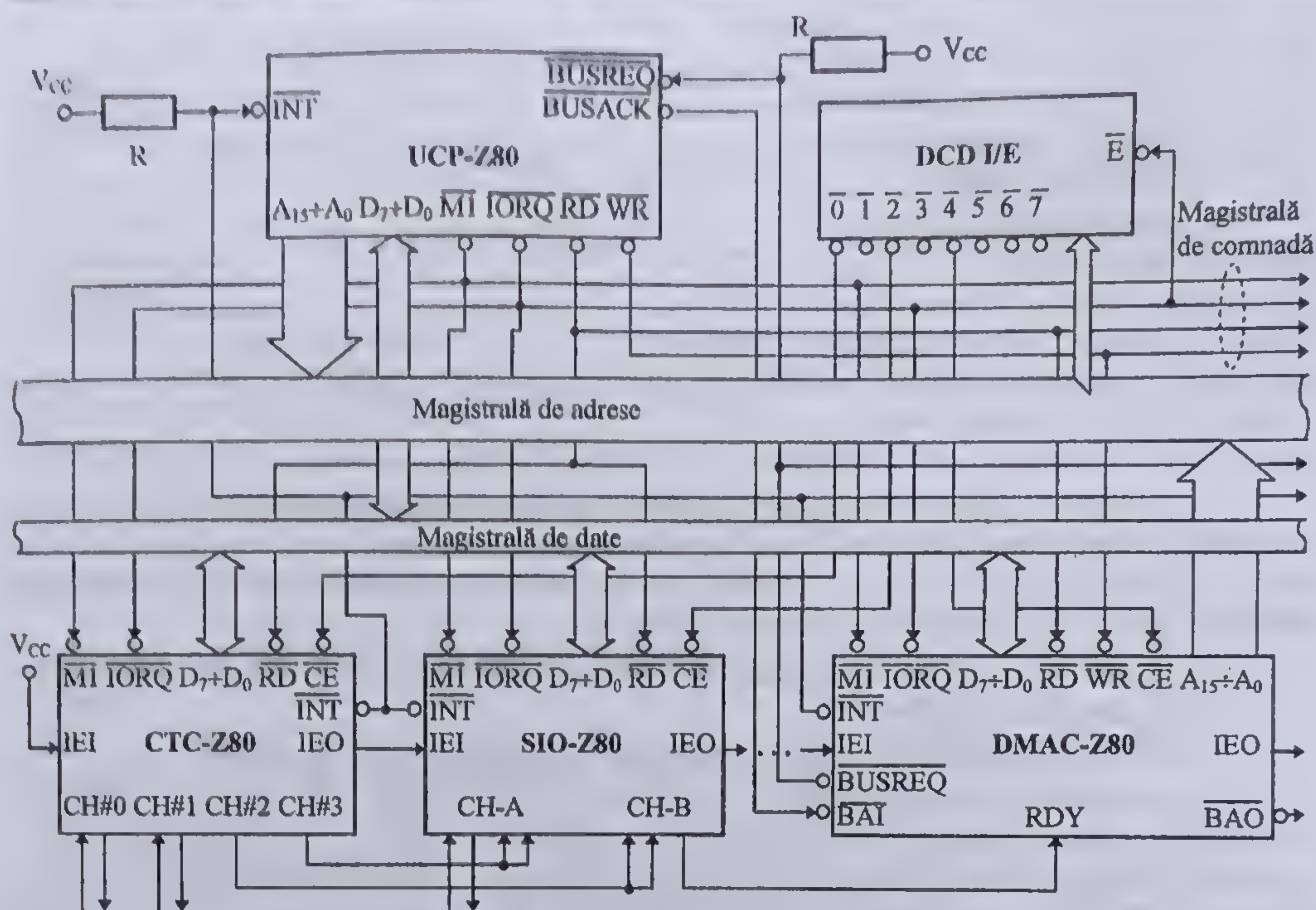


Fig.5.22. Conectarea la magistralele sistemului a circuitelor din familia Z80

5.2. Programarea microprocesorului Z80

Fiind conceput ca o dezvoltare și o perfecționare a procesorului 8080A, programarea microprocesorului Z80 comportă reguli și principii similare. Dar, datorită extinderii numărului de registre și a setului de instrucțiuni, există deosebiri în ceea ce privește formatul instrucțiunilor și modurile de adresare, diferențe ce vor fi evidențiate în continuare.

5.2.1. Formatul instrucțiunilor și modurile de adresare

Formatul instrucțiunilor este și în acest caz variabil și poate fi compus din 1+4 octeți (fig.5.23). Codul operației este la Z80 de unul sau doi octeți, pentru a se putea reprezenta cele 696 de coduri distincte, corespunzătoare celor 158 de tipuri de instrucțiuni.

Instrucțiunile microprocesorului 8080A (în număr de 244), care au codul operației de un octet, se regăsesc integral și la Z80. Din cele 12 combinații neutilizate de 8080, 8 sunt opcoduri pentru instrucțiuni noi, tot de un singur octet, iar restul de 4 stau la baza a tot atâtea grupuri de instrucțiuni cu opcodul de doi octeți. Ca și la procesoarele Intel, după codul operației pot urma, eventual, unul sau doi octeți (data/adresa).

În ceea ce privește modurile de adresare, microprocesorul Z80 utilizează trei moduri noi, în plus față de cele cinci moduri de bază ale familiei 8080/8085. Astfel, modurile de adresare ale microprocesorului Z80 sunt:

- **adresarea implicită**, la instrucțiuni care acționează asupra unor operanzi amplasați numai într-o anumită locație (de exemplu registrul acumulator este implicit și destinația rezultatului, la majoritatea operațiilor aritmetice și logice pe 8 biți).

- **adresarea explicită** (la registru), apare la instrucțiunile pentru care codul operație indică în mod explicit registrele interne implicate în realizarea operației.

- **adresarea imediată**, când operandul este conținut în corpul instrucțiunii, imediat după codul operație. În cazul în care operandul este de doi octeți, acest mod se mai numește, la Z80, *adresare imediată extinsă*. În ambele cazuri, adresa operanzilor se obține prin simpla incrementare a registrului PC.

- **adresarea directă**, când în corpul instrucțiunii se află adresa operandului, se consideră a fi, la Z80, de două tipuri: *adresare directă în pagina zero* - corespunzătoare instrucțiunilor de tip RST p , respectiv *adresare directă extinsă*, când adresa este specificată complet, printr-unul (adresă de port I/E) sau doi octeți (adresă de memorie).

- **adresarea indirectă**, se realizează cu unul dintre registrele de 16 biți ale microprocesorului, care conține adresa operandului.

Urmează *modurile de adresare specifice* microprocesorului Z80, și anume:

- **adresarea indexată**, care este o adresare de tip indirect, folosind registrele index IX sau IY. Adresa operandului se calculează prin adunarea, la adresa conținută de IX sau IY (adresa de bază), a unei constante de un octet, numită *deplasament* (d). Acesta este amplasat în memorie imediat după codul operației și este considerat ca un întreg cu semn, în complement față de 2, pe 8 biți ($d \in [-128, 127]$). Deplasamentul poate fi privit ca un *index* în raport cu adresa de bază, conținută de IX sau IY.

La Z80, instrucțiunile care folosesc adresarea indexată sunt de trei sau patru octeți, dintre care primii doi formează codul operație (fig.5.24).

Exemple:

1) **ADD A, (IX+d)** ; $A \leftarrow A + (IX+d)$: se adună la acumulator conținutul locației de memorie a cărei adresă este dată de suma $IX+d$, iar rezultatul se păstrează în A. Codul obiect al instrucțiunii este {DDh, 86h, d } (v.fig.5.24).

2) **INC (IY+d)** ; $(IY+d) \leftarrow [(IY+d) + 1]$: se incrementează cu 1 octetul din memorie, de la adresa dată de suma $IY+d$. Codul obiect al instrucțiunii, în hexazecimal, este {FDh, 34h, d }.

Introducerea registrelor IX și IY și a adresării indexate a determinat extinderea unor tipuri de instrucțiuni, care la 8080/8085 se limitează numai la utilizarea registrului pereche H (HL). *Combinarea* adresării indexate cu alte tipuri, de cele mai multe ori cu adresarea imediată sau la registru, crește versatilitatea instrucțiunilor microprocesorului Z80.

3) **LD (IX+d), n** ; $(IX+d) \leftarrow n$: octetul n se încarcă (LoaD) în memorie, la adresa dată de suma $IX+d$. Codul obiect are 4 octeți, cu doi operanzi: {DDh, 36h, d , n }. Se utilizează adresările indexată și imediată.

4) **LD r, (IY+d)** ; $r \leftarrow (IY+d)$: conținutul locației de memorie specificate prin $IY+d$ se transferă în registrul r (A, B, C, D, E, H sau L). Codul obiect include în al doilea octet al opcodului adresa (codul) de trei biți a registrului r (adresare la registru).

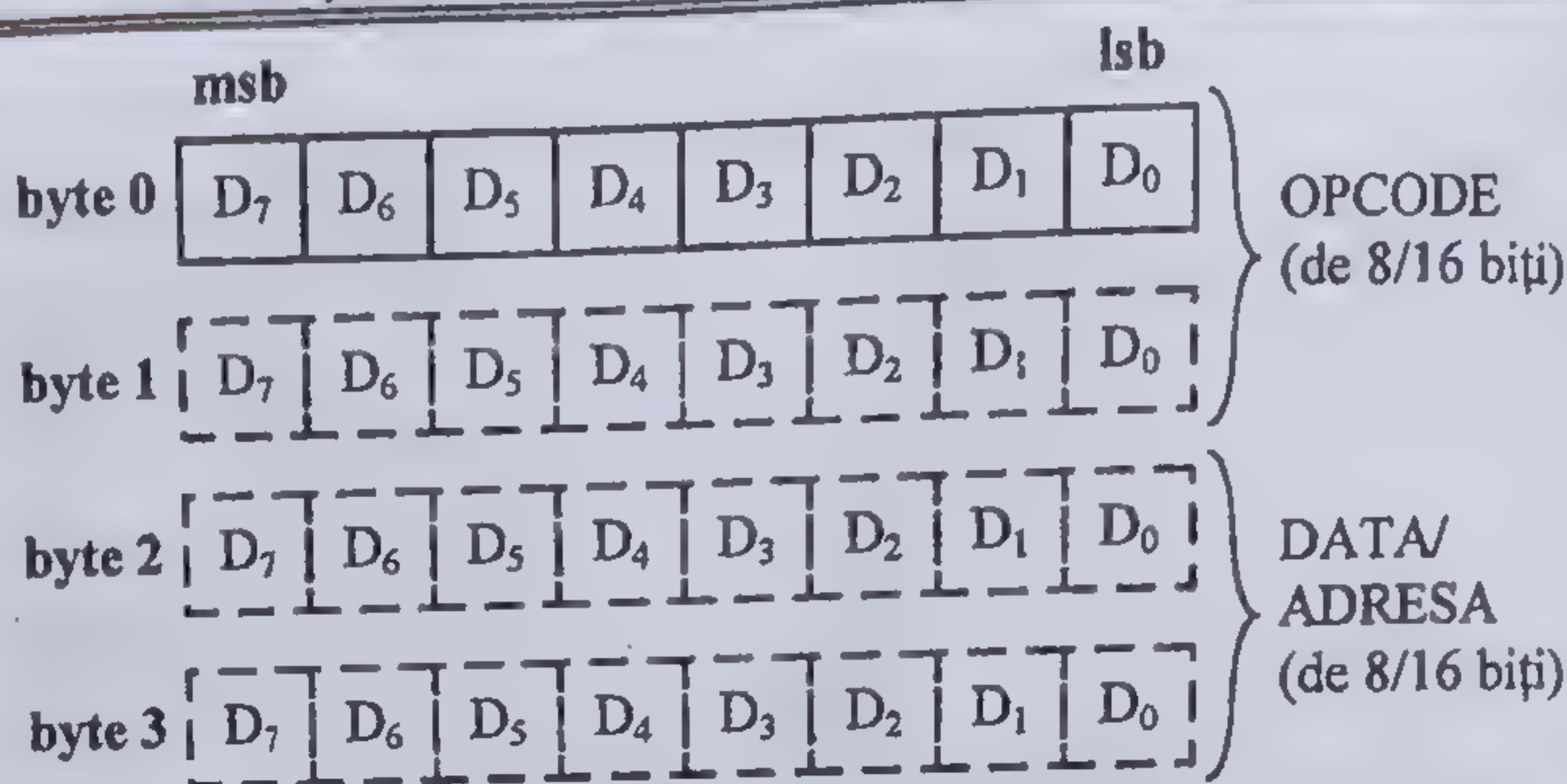


Fig.5.23. Formatul instrucțiunilor la microprocesorul Z80

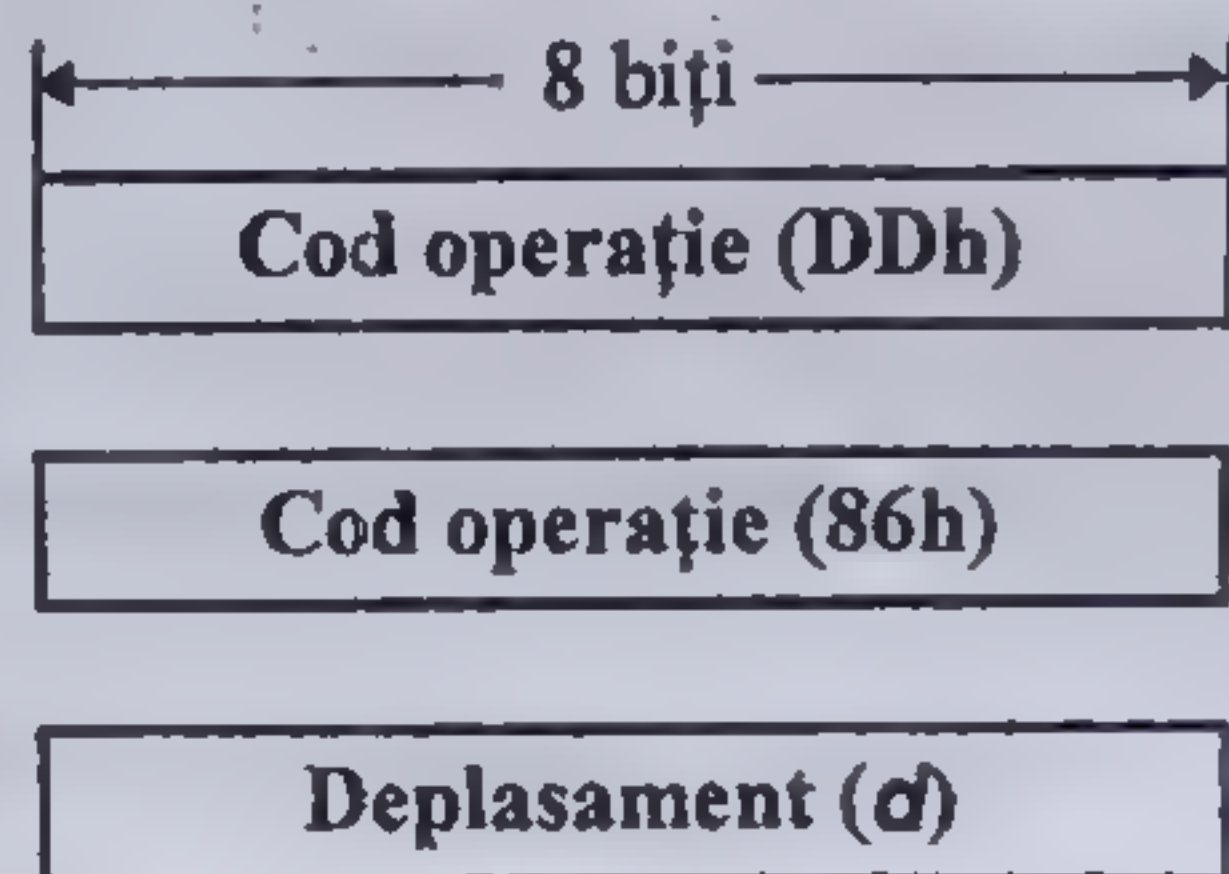


Fig.5.24. Formatul instrucțiunilor cu adresare indexată.
Ex.: ADD A, (IX+d)

Partea a II-a - Sisteme cu microprocesoare de 8 biți

- **adresarea relativă** (vezi și §1.1.1.4) permite formarea adresei operandului prin adunarea la adresa instrucțiunii curente (\$), a unei expresii e , cu valoarea cuprinsă între -126 și +129. La momentul execuției, microprocesorul extrage mai întâi codul instrucțiunii, de doi octeți, registrul PC fiind poziționat pe instrucțiunea următoare. Cel de-al doilea octet al instrucțiunii, interpretat ca un întreg în complement față de 2 (în gama [-128, 127]), se adună la conținutul din acel moment al registrului PC ($\$+2$). Astfel, instrucțiunile care folosesc adresarea relativă pot să efectueze ramificații în interiorul unui bloc de 256 de octeți, centrat pe adresa instrucțiunii următoare.

S-a constatat că majoritatea instrucțiunilor de salt se execută peste un număr mic de instrucțiuni, ceea ce se implementează mai avantajos printr-o adresare relativă (2 octeți) decât printr-o adresare absolută (3 octeți).

Exemple:

1) JR e ; PC $\leftarrow \$+e$: salt relativ necondiționat, cu valoarea e .

2) JR c, e ; Dacă este îndeplinită condiția c , PC $\leftarrow \$+e$: salt relativ condiționat, unde c poate fi C, NC, Z sau NZ; altfel, PC $\leftarrow \$+2$.

Prin combinarea adresării relative cu decrementarea registrului B s-a obținut o instrucțiune scurtă de buclare, deosebit de avantajoasă pentru programarea ciclurilor cu contor de un octet:

3) DJNZ e ; B $\leftarrow B-1$; dacă $B \neq 0$, PC $\leftarrow \$+e$: salt relativ condiționat, cât timp conținutul registrului B, decrementat, rămâne diferit de 0; dacă $B=0$, atunci PC $\leftarrow \$+2$.

- **adresarea pe bit** permite referirea directă la oricare din biții unui octet, în vederea setării, resetării sau testării valorii acestuia. Se evită astfel folosirea operațiilor logice cu mascare sau deplasări în fanionul de transport, utilizate pentru manipulări la nivel de bit la procesoarele 8080/8085.

Exemple:

1) BIT b, r ; $Z \leftarrow r_b$: complementul bitului b ($0 \div 7$) al registrului r (A, B, C, D, E, H, L) se transferă în fanionul Z. Atât b , cât și r , sunt specificați direct, în cel de-al doilea octet al codului operației. Aducerea bitului în Z permite luarea unei decizii (ramificare condiționată de Z sau NZ).

2) SET $b, (IX+d)$; $(IX+d)_b \leftarrow 1$: bitul b al octetului din memorie, de la adresa dată de suma $IX+d$, este forțat în "1" logic.

3) RES b, r ; $r_b \leftarrow 0$: bitul b al registrului r , specificat în instrucțiune, este forțat în "0" logic.

Instrucțiunile cu adresare pe bit sunt avantajoase pentru implementarea de automate programabile pe bit și, în general, în realizarea structurilor logice de automatizare.

5.2.2. Setul de instrucțiuni

Setul de instrucțiuni al microprocesorului Z80 este unul dintre cele mai puternice în raport cu celelalte tipuri de microprocesoare de 8 biți.

După cum s-a menționat mai înainte, Z80 este compatibil "în jos" cu 8080/8085 (cu excepția instrucțiunilor SIM și RIM de la 8085), adică instrucțiunile acestora (78) se regăsesc cu același cod obiect și la Z80. În schimb, mărirea substanțială a repertoriului de instrucțiuni (la 158) și introducerea unor noi moduri de adresare au impus regândirea formatului extern al instrucțiunilor. Cu această ocazie s-au modificat mnemonicele pentru majoritatea instrucțiunilor, precum și modul de specificare a adresării indirecte a operanzilor.

Codul mnemonic al unei instrucțiuni, la Z80, este determinat numai de funcția realizată, spre deosebire de cazul microprocesoarelor Intel, la care s-a ținut cont și de alți factori, precum modul de adresare, lungimea operandului sau numele unor registre. În acest fel a fost posibilă

reducerea numărului de mnemonice de instrucțiuni și, drept urmare, o simplificare a codificării și citirii programelor în limbaj de asamblare. Spre exemplu, în locul mnemonicelor MOV, MVI, LDA, LHLD, LDAX, LXI, STA, SHLD, STAX de la 8080/8085, la Z80 se utilizează numai mnemonica LD, pentru orice transfer de octeți sau cuvinte, indiferent de modul de adresare și de registrele utilizate. De asemenea, pentru instrucțiunile des utilizate s-a apelat la coduri scurte, de numai două litere: LD, CP (în loc de CMP și CPI), JP (în loc de JMP) sau JR.

Astfel, se poate afirma că formatul Zilog al instrucțiunilor în limbaj de asamblare este superior formatului Intel, deoarece este mai explicit în referirea operanzilor [30, 32].

Pentru o mai bună înțelegere a setului de instrucțiuni și a folosirii acestuia la scrierea programelor în limbaj de asamblare, se vor evidenția convențiile utilizate în codul mnemonic Zilog.

1) Numele unui registru pereche este format din ambele nume ale registrelor componente. Cuvântul de stare al programului, format din registrul A și indicatorii de condiție (Flags), se notează tot cu două litere: AF (AF'), în loc de PSW.

Exemple:

ADD	HL,BC	; echivalent cu DAD B (la 8080/8085)
PUSH	DE	; echivalent cu PUSH D
LD	SP,IY	
POP	AF	; în loc de POP PSW

2) Adresarea indirectă prin HL folosește chiar numele registrului pereche în locul literei M de la 8080/8085. În general, numele unui registru de 16 biți între paranteze indică o adresare indirectă (sau indexată) cu registrul respectiv.

Exemple:

LD	B,(HL)	; echivalent cu MOV B,M
DEC	(IX+9)	
LD	(BC),A	; echivalent cu STAX B
LD	SP,HL	; echivalent cu SPHL

3) O constantă numerică sau simbolică fără paranteze indică un operand imediat (se folosește valoarea operandului).

Exemple:

LD	B,0C3h	; echivalent cu MVI B,0C3h
LD	HL,val	; echivalent cu LXI H,val

4) O constantă numerică sau simbolică în paranteze, considerată pe 16 biți, indică adresa de memorie a operandului și se utilizează în instrucțiunile cu adresare directă a memoriei. În cazul porturilor I/E, constanta din paranteză este pe 8 biți și indică adresa portului utilizat.

Exemple:

LD	(3FFh),A	; echivalent cu STA 3FFh
LD	HL,(adr)	; echivalent cu LHLD adr
LD	(tab),IX	; (tab) ← IX _L , (tab+1) ← IX _H
IN	A,(40h)	; echivalent cu IN 40h
OUT	(CTC),A	; echivalent cu OUT CTC

5) Condițiile de ramificație se constituie ca prim operand în instrucțiunilor de salt, de apel și de revenire, care au mnemonice unice: JP, CALL, RET.

Exemple:

JP	C,et1	; echivalent cu JC et1
JP	NZ,et2	; echivalent cu JNZ et2
CALL	NC,rut1	; echivalent cu CNC rut1
CALL	PE,rut2	; echivalent cu CPE rut2
RET	Z	; echivalent cu RZ

RET PO ; echivalent cu RPO

6) Registrul A apare explicit în instrucțiunile aritmetice, ca operand, chiar dacă operandul este localizat implicit în acumulator, spre deosebire de instrucțiunile de prelucrare logică, când numele acumulatorului este omis:

Exemple:

ADD A,B ; echivalent cu ADD B
CP C ; echivalent cu CMP C
XOR (HL) ; echivalent cu XRA M

În continuare se prezintă setul de instrucțiuni al microprocesorului Z80 (tabelele 5.3÷5.8), clasificat după criteriul funcționalității, ca și la 8080/8085 (v. §4.3.3). Se specifică mnemonicele Z80, împreună cu mnemonicele echivalente de la 8080 (dacă există), descrierea simbolică a instrucțiunilor, indicatorii de condiție afectați, precum și numărul de cicluri mașină și numărul de stări necesare pentru execuție. Informații suplimentare se pot obține din [10, 27, 28, 32, 44, 45].

Se menționează faptul că, în afara notațiilor folosite până în acest moment, se vor mai utiliza și următoarele:

n - expresie de un octet, în gama [0, 255];

nn - expresie de doi octeți, în gama [0, 65535].

5.2.2.1. Instrucțiuni de transfer

La acest grup de instrucțiuni se poate evidenția creșterea numărului de instrucțiuni de transfer pe 16 biți față de 8080 (de la 3 la 11) și introducerea *transferurilor pe blocuri de octeți*.

Tab.5.3.

Mnemonica Z80	Mnemonica 8080	Descrierea operației	Indicatorii afectați	Nr. CM/ nr. stări	Comentarii
Transferuri pe 8 biți					
LD r, r'	MOV r, r'	$r \leftarrow r'$	-	1/4	Cod r, r'
LD r, n	MVI r, n	$r \leftarrow n$	-	2/7	000 B
LD $r, (HL)$	MOV r, M	$r \leftarrow (HL)$	-	2/7	001 C
LD $r, (IX+d)$		$r \leftarrow (IX+d)$	-	5/9	010 D
LD $r, (IY+d)$		$r \leftarrow (IY+d)$	-	5/9	011 E
LD $(HL), r$	MOV M, r	$(HL) \leftarrow r$	-	2/7	100 H
LD $(IX+d), r$		$(IX+d) \leftarrow r$	-	5/19	101 L
LD $(IY+d), r$		$(IY+d) \leftarrow r$	-	5/19	111 A
LD $(HL), n$	MVI M, n	$(HL) \leftarrow n$	-	3/10	
LD $(IX+d), n$		$(IX+d) \leftarrow n$	-	5/19	
LD $(IY+d), n$		$(IY+d) \leftarrow n$	-	5/19	
LD $A, (BC)$	LDAX B	$A \leftarrow (BC)$	-	2/7	
LD $A, (DE)$	LDAX D	$A \leftarrow (DE)$	-	2/7	
LD $A, (nn)$	LDA nn	$A \leftarrow (nn)$	-	4/13	
LD $(BC), A$	STAX B	$(BC) \leftarrow A$	-	2/7	
LD $(DE), A$	STAX D	$(DE) \leftarrow A$	-	2/7	
LD $(nn), A$	STA nn	$(nn) \leftarrow A$	-	4/13	
LD A, I		$A \leftarrow I$	S, Z, P/V=IFF ₁ , H=N=0	2/9	
LD A, R		$A \leftarrow R$	Idem	2/9	
LD I, A		$I \leftarrow A$	-	2/9	
LD R, A		$R \leftarrow A$	-	2/9	

Tab.5.3 (cont.)

Transferuri pe 16 biți					
LD <i>dd,nn</i>	LXI <i>rp,nn</i>	$dd \leftarrow nn$	-	3/10	<div>Cod <i>dd</i></div> <div>00 BC</div> <div>01 DE</div> <div>10 HL</div> <div>11 SP</div>
LD IX, <i>nn</i>		$IX \leftarrow nn$	-	4/14	
LD IY, <i>nn</i>		$IY \leftarrow nn$	-	4/14	
LD HL,(<i>nn</i>)	LHLD <i>nn</i>	$L \leftarrow (nn), H \leftarrow (nn+1)$	-	5/16	
LD <i>dd,(nn)</i>		$dd_L \leftarrow (nn), dd_H \leftarrow (nn+1)$	-	6/20	
LD IX,(<i>nn</i>)		$IX_L \leftarrow (nn), IX_H \leftarrow (nn+1)$	-	6/20	
LD IY,(<i>nn</i>)		$IY_L \leftarrow (nn), IY_H \leftarrow (nn+1)$	-	6/20	
LD (<i>nn</i>),HL	SHLD <i>nn</i>	$(nn) \leftarrow L, (nn+1) \leftarrow H$	-	5/16	
LD (<i>nn</i>), <i>dd</i>		$(nn) \leftarrow dd_L, (nn+1) \leftarrow dd_H$	-	6/20	
LD (<i>nn</i>),IX		$(nn) \leftarrow IX_L, (nn+1) \leftarrow IX_H$	-	6/20	
LD (<i>nn</i>),IY		$(nn) \leftarrow IY_L, (nn+1) \leftarrow IY_H$	-	6/20	
Schimburi între registre și transfer pe blocuri					
EX DE,HL	XCHG	$DE \leftrightarrow HL$	-	1/4	
EX AF,AF'		$AF \leftrightarrow AF'$	-	1/4	
EXX		$BC \leftrightarrow BC'$ $DE \leftrightarrow DE'$ $HL \leftrightarrow HL'$	-	1/4	
LDI (Load and Increment)		$(DE) \leftarrow (HL)$ $DE \leftarrow DE+1$ $HL \leftarrow HL+1$ $BC \leftarrow BC-1$	H=N=0, iar P/V=0 dacă BC=0 după decrementare	4/16	
LDIR (Load, Increment and Repeat)		$(DE) \leftarrow (HL)$ $DE \leftarrow DE+1$ $HL \leftarrow HL+1$ $BC \leftarrow BC-1$	se repetă până când BC=0	4/16, dacă BC=0 5/21, dacă BC≠0	
LDD (Load and Decrement)		$(DE) \leftarrow (HL)$ $DE \leftarrow DE-1$ $HL \leftarrow HL-1$ $BC \leftarrow BC-1$	La fel ca LDI		
LDDR (Load, Decrement and Repeat)		$(DE) \leftarrow (HL)$ $DE \leftarrow DE-1$ $HL \leftarrow HL-1$ $BC \leftarrow BC-1$	se repetă până când BC=0	La fel ca LDIR	

Ultima categorie asigură realizarea unor operații complexe, care la procesoarele Intel se realizează prin subprograme, cu un consum mult mai mare de memorie și de timp. Spre exemplu, instrucțiunile de transfer cu repetiție (LDIR, LDDR) permit transferuri de blocuri de până la 64 Kocteți, dintr-o zonă de memorie în alta, după precizarea adreselor sursă (în HL) și destinație (în DE), respectiv a dimensiunii blocului (în BC). Zonele de memorie se pot și suprapune, iar transferul datelor se face fără intervenția acumulatorului.

5.2.2.2. Instrucțiuni aritmetice

Se poate remarca și în acest caz creșterea numărului și tipurilor de operații pe 16 biți, față de cele realizate cu procesoarele 8080/8085. Spre exemplu, numărul instrucțiunilor de operare aritmetică propriu-zisă (adunare-scădere) se diversifică și crește de 5 ori, de la 4 (DAD *rp*) la 20 (ADD, ADC, SBC).

Tab.5.4.

Mnemonica Z80	Mnemonica 8080	Descrierea operației	Indicatorii afectați	Nr. CM/ nr. stări	Comentarii
Operații aritmetice pe 8 biți					
ADD A,r	ADD r	$A \leftarrow A+r$	S,Z,H,P/V,C, N=0	1/4	Cod r.
ADD A,n	ADI n	$A \leftarrow A+n$	idem	2/7	000 B
ADD A,(HL)	ADD M	$A \leftarrow A+(HL)$	idem	2/7	001 C
ADD A,(IX+d)		$A \leftarrow A+(IX+d)$	idem	5/19	011 E
ADD A,(IY+d)		$A \leftarrow A+(IY+d)$	idem	5/19	100 H 101 L 111 A
ADC A,s	ADC r ADC M ACI n	$A \leftarrow A+s+CY$	idem		s - poate fi r, n, (HL), (IX+d) sau (IY+d), ca la ADD.
SUB s	SBB r SBB M SBI n	$A \leftarrow A-s$	idem		
SBC A,s	SBB r SBB M SBI n	$A \leftarrow A-s-CY$	idem		
INC r	INR r	$r \leftarrow r+1$	S,Z,H,P/V, N=0	1/4	
INC (HL)	INR M	$(HL) \leftarrow (HL)+1$	idem	3/11	
INC (IX+d)		$(IX+d) \leftarrow (IX+d)+1$	idem	6/23	
INC (IY+d)		$(IY+d) \leftarrow (IY+d)+1$	idem	6/23	
DEC m	DCR r DCR M	$m \leftarrow m-1$	S, Z, H, P/V, N=1		m - poate fi r, (HL), (IX+d) sau (IY+d), ca la INC.
DAA	DAA		S, Z, H, P/V, C	1/4	
Operații aritmetice pe 16 biți					
ADD HL,ss	DAD rp	$HL \leftarrow HL+ss$	C, N=0	3/11	Cod ss
ADC HL,ss		$HL \leftarrow HL+ss+CY$	S, Z, P/V, C, N=0	4/15	00 BC
SBC HL,ss		$HL \leftarrow HL-ss-CY$	S, Z, P/V, C, N=1	4/15	01 DE 10 HL 11 SP
ADD IX,pp		$IX \leftarrow IX+pp$	C, N=0	4/15	Cod pp
ADD IY,rr		$IY \leftarrow IY+rr$	idem	4/15	00 BC
INC ss	INR rp	$ss \leftarrow ss+1$	-	1/6	01 DE
INC IX		$IX \leftarrow IX+1$	-	2/10	10 IX
INC IY		$IY \leftarrow IY+1$	-	2/10	11 SP
DEC ss	DCX rp	$ss \leftarrow ss-1$	-	1/6	Cod rr
DEC IX		$IX \leftarrow IX-1$	-	2/10	00 BC 01 DE
DEC IY		$IY \leftarrow IY-1$	-	2/10	10 IY 11 SP

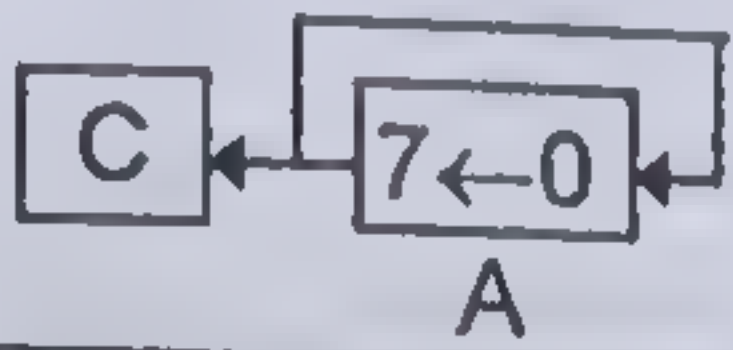
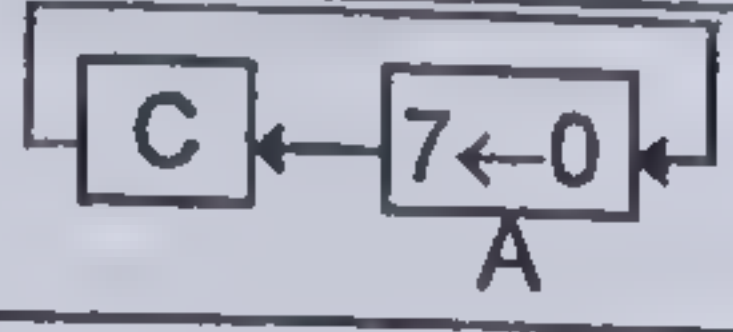
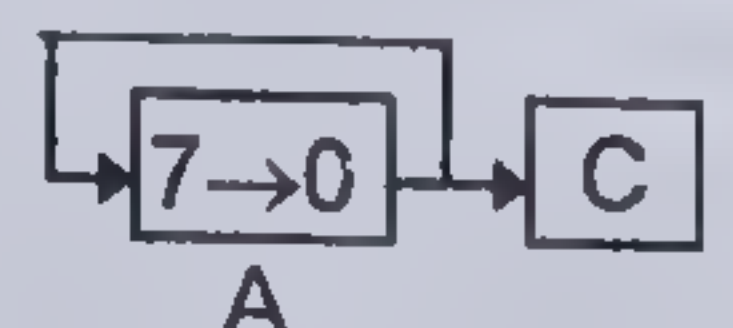
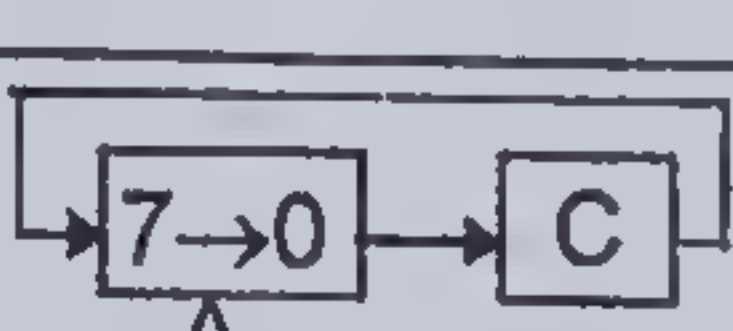
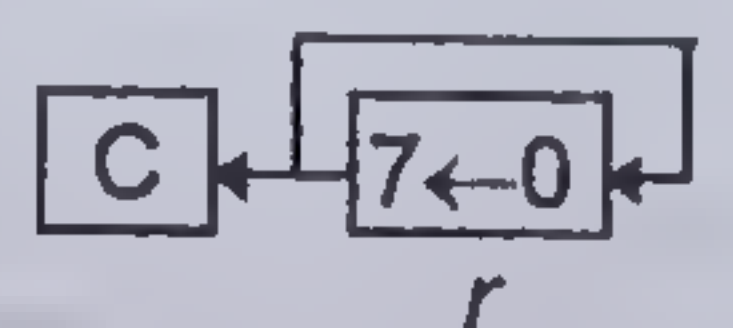
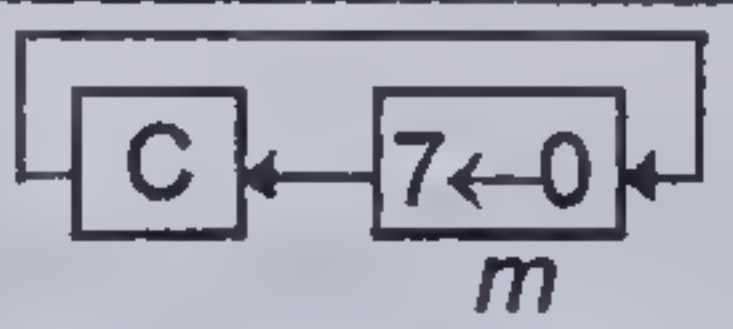
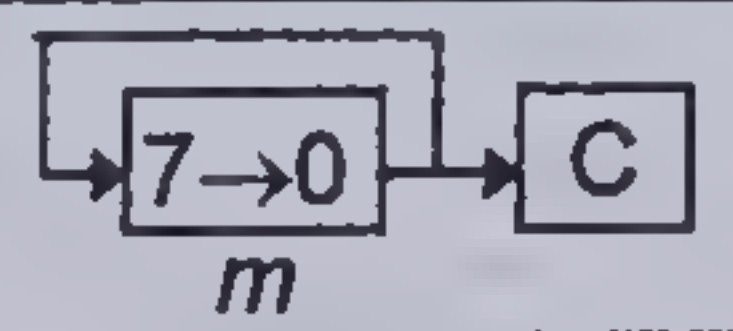
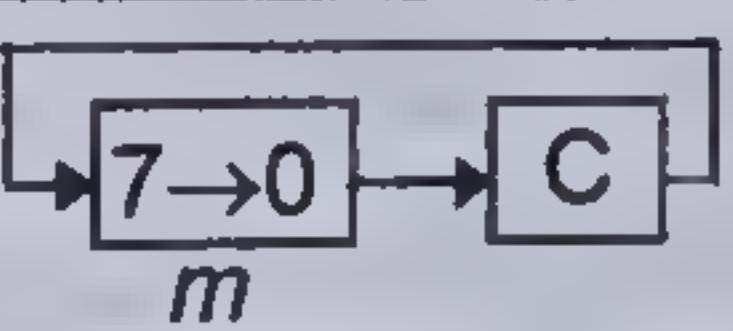
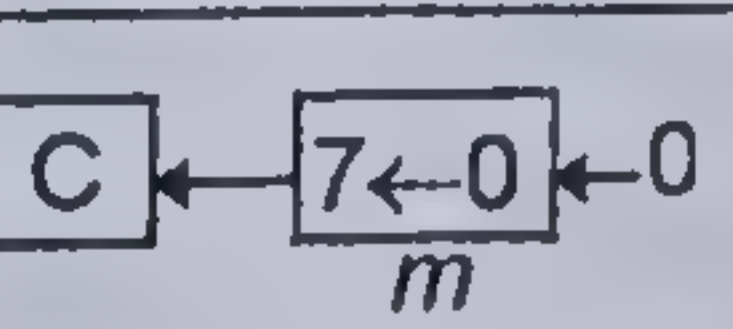
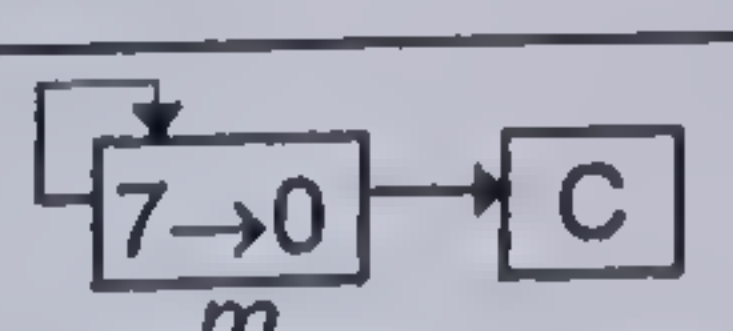
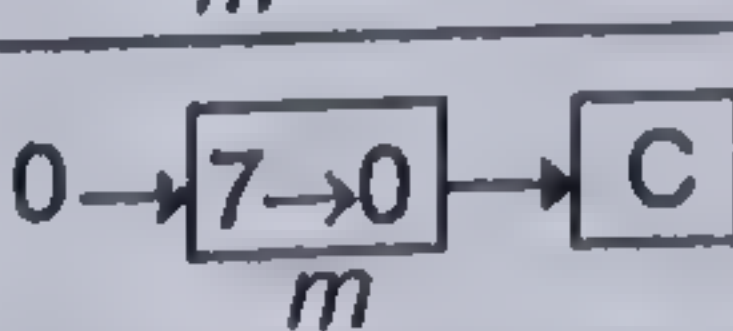
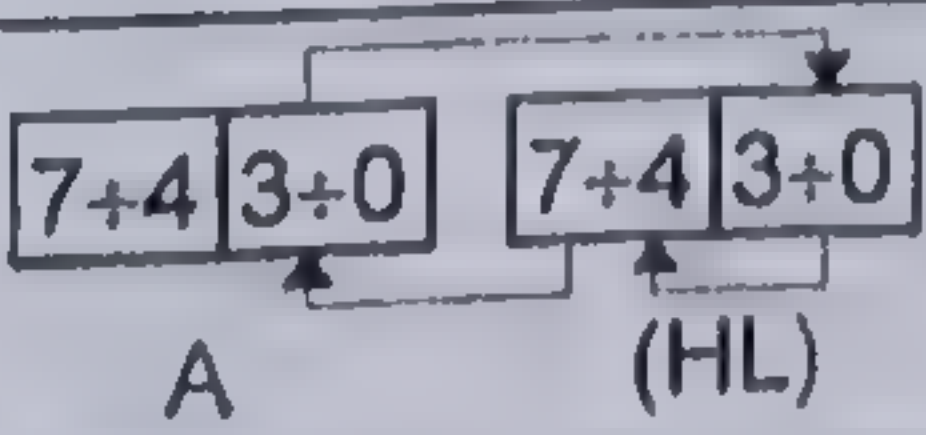
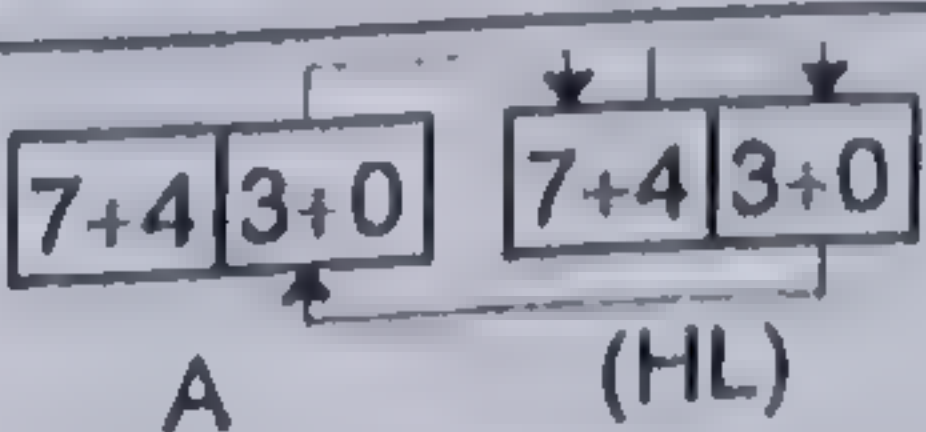
5.2.2.3. Instrucțiuni logice

Acest grup de instrucțiuni a fost mult îmbunătățit în ceea ce privește prelucrarea pe octet și semioctet (digit), dar mai ales la nivel de bit. În primul rând se evidențiază *instrucțiunile de căutare* într-un bloc de memorie, cu sau fără repetiție (CPIR, CPDR, CPI, CPD), care sunt o extensie importantă a instrucțiunilor de comparare de la 8080.

Tab.5.5.

Mnemonica Z80	Mnemonica 8080	Descrierea operației	Indicatorii afectați	Nr. CM/nr. stări	Comentarii
Operații logice					
AND <i>r</i>	ANA <i>r</i>	$A \leftarrow A \wedge r$	S,Z,P/V, C=H=N=0	1/4	Cod <i>r</i> 000 B
AND <i>n</i>	ANI <i>n</i>	$A \leftarrow A \wedge n$	idem	2/7	001 C
AND (HL)	ANA M	$A \leftarrow A \wedge (HL)$	idem	2/7	011 E
AND (IX+d)		$A \leftarrow A \wedge (IX+d)$	idem	5/19	100 H
AND (IY+d)		$A \leftarrow A \wedge (IY+d)$	idem	5/19	101 L 111 A
OR <i>s</i>	ORA <i>r</i> ORA M ORI <i>n</i>	$A \leftarrow A \vee s$	idem		<i>s</i> - poate fi <i>r</i> , <i>n</i> , (HL), (IX+d) sau (IY+d), ca la AND.
XOR <i>s</i>	XRA <i>r</i> XRA M XRI <i>n</i>	$A \leftarrow A \oplus s$	idem		
CPL	CMA	$A \leftarrow \overline{A}$	H=N=1	1/4	Complement față de 1
NEG		$A \leftarrow 0 - A$	S,Z,H,P/V,C, N=1	2/8	Complement față de 2
CCF	CMC	$CY \leftarrow \overline{CY}$	C, N=0	1/4	
SCF	STC	$CY \leftarrow 1$	C=1, H=N=0	1/4	
Comparări și căutări pe bloc					
CP <i>s</i>	CMP <i>r</i> CMP M CPI <i>n</i>	$A - s$	S,Z,H,P/V,C, N=1		
CPI (Compare and Increment)		$A - (HL)$ $HL \leftarrow HL + 1$ $BC \leftarrow BC - 1$	S,H,N=1, Z=1 dacă $A=(HL)$, P/V=0 dacă $BC=0$ după decrementare	4/16	
CPIR (Compare, Increment and Repeat)		$A - (HL)$ $HL \leftarrow HL + 1$ $BC \leftarrow BC - 1$ } se repetă până când $A=(HL)$ sau $BC=0$	idem	4/16 - BC≠0, 5/21 - BC=0	
CPD (Compare and Decrement)		$A - (HL)$ $HL \leftarrow HL - 1$ $BC \leftarrow BC - 1$	La fel ca la CPI		
CPDR (Compare Decrement and Repeat)		$A - (HL)$ $HL \leftarrow HL - 1$ $BC \leftarrow BC - 1$ } se repetă până când $A=(HL)$ sau $BC=0$	La fel ca la CPIR		

Tab.5.5 (cont.)

Rotiri și deplasări					
RLCA (Rotate Left Circular Accumulator)	RLC		C, H=N=0	1/4	
RLA (Rotate Left Accumulator)	RAL		C, H=N=0	1/4	
RRCA (Rotate Right Circular Accumulator)	RRC		idem	1/4	
RRA (Rotate Right Accumulator)	RAR		idem	1/4	
RLC <i>r</i>			S,Z,P,C, H=N=0	2/8	
RLC (HL)		<i>r</i>	idem	4/15	
RLC (IX+d)		(HL)	idem	6/23	
RLC (IY+d)		(IX+d)	idem	6/23	
RL <i>m</i>			idem		<i>m</i> - poate fi <i>r</i> , (HL), (IX+d) sau (IY+d), ca la RLC.
RRC <i>m</i>			idem		
RR <i>m</i>			idem		
SLA <i>m</i> (Shift Left Arithmetic)			idem		
SRA <i>m</i>			idem		
SRL <i>m</i> (Shift Right Logic)			idem		
RLD (Rotate Left Digit)			idem	5/18	
RRD (Rotate Right Digit)			idem	5/18	

Tab.5.5 (cont.)

Prelucrări pe bit					
BIT b,r		$Z \leftarrow \overline{r_b}$	$Z, H=1, N=0$	2/8	Cod b
BIT $b,(HL)$		$Z \leftarrow \overline{(HL)_b}$	idem	3/12	000 0
BIT $b,(IX+d)$		$Z \leftarrow \overline{(IX+d)_b}$	idem	5/20	001 1
BIT $b,(IY+d)$		$Z \leftarrow \overline{(IY+d)_b}$	idem	5/20	010 2
SET b,r		$r_b \leftarrow 1$	-	2/8	011 3
SET $b,(HL)$		$(HL)_b \leftarrow 1$	-	4/15	100 4
SET $b,(IX+d)$		$(IX+d)_b \leftarrow 1$	-	6/23	110 6
SET $b,(IY+d)$		$(IY+d)_b \leftarrow 1$	-	6/23	111 7
RES b,m		$m_b \leftarrow 0$	-		m - poate fi $r, (HL), (IX+d)$ sau $(IY+d)$, ca la SET.

Se poate observa creșterea substanțială a numărului de instrucțiuni de rotire, dar mai ales apariția instrucțiunilor noi de deplasare și de prelucrare pe bit.

5.2.2.4. Instrucțiuni de ramificare

În acest grup de instrucțiuni se remarcă instrucțiunile care utilizează adresarea relativă și care asigură o scriere mai compactă a programelor ce conțin salturi scurte.

Tab.5.6.

Mnemonica Z80	Mnemonica 8080	Descrierea operației	Indicatorii afectați	Nr. CM/nr. stări	Comentarii
Instrucțiuni de salt					
JP nn	JMP nn	$PC \leftarrow nn$	-	3/10	Cod cc
JP cc,nn	Jcc nn	Dacă cc e adevărată, $PC \leftarrow nn$; altfel continuă	-	3/10	000 NZ (non-zero)
JR e		$PC \leftarrow PC+e$	-	3/12	001 Z (zero)
JR C,e		Dacă $C=1$, $PC \leftarrow PC+e$; altfel, continuă.	-	3/12 2/7	010 NC (non-carry)
JR NC,e		Dacă $C=0$, $PC \leftarrow PC+e$; altfel, continuă.	-	3/12 2/7	011 C (carry)
JR Z,e		Dacă $Z=1$, $PC \leftarrow PC+e$; altfel, continuă.	-	3/12 2/7	100 PO (parity odd)
JR NZ,e		Dacă $Z=0$, $PC \leftarrow PC+e$; altfel, continuă.	-	3/12 2/7	101 PE (even)
JP (HL)	PCHL	$PC \leftarrow HL$	-	1/4	110 P (plus)
JP (IX)		$PC \leftarrow IX$	-	2/8	111 M (minus)
JP (IY)		$PC \leftarrow IY$	-	2/8	
DJNZ e (Decrement and Jump if Non-Zero)		$B \leftarrow B-1$ Dacă $B \neq 0$ după decrementare, $PC \leftarrow PC+e$; altfel, continuă.	-	3/13 2/8	

Tab.5.6 (cont.)

Apeluri de subrutine și reveniri					
CALL <i>nn</i>	CALL <i>nn</i>	$SP \leftarrow SP-1, (SP) \leftarrow PC_H$ $SP \leftarrow SP-1, (SP) \leftarrow PC_L$ $PC \leftarrow nn$	-	5/17	<div>7 38h</div> <div>6 30h</div> <div>5 28h</div> <div>4 20h</div> <div>3 18h</div> <div>2 10h</div> <div>1 08h</div> <div>0 00h</div> <div><i>t</i> <i>p</i></div>
CALL <i>cc,nn</i>	Ccc <i>nn</i>	Dacă <i>cc</i> e adevărată, CALL <i>nn</i> ; altfel, continuă.	-	5/17 3/10	
RET	RET	$PC_L \leftarrow (SP),$ $SP \leftarrow SP+1, PC_H \leftarrow (SP)$ $SP \leftarrow SP+1$	-	3/10	
RET <i>cc</i>	Rcc	Dacă <i>cc</i> e adevărată, RET ; altfel, continuă.	-	3/11 1/5	
RETI (Return from Interrupt)		RET din întrerupere	-	4/14	
RETN (Return from NMI)		$IFF_1 \leftarrow IFF_2,$ RET din întreruperea nemascabilă	-	4/14	
RST <i>p</i>	RST <i>t</i>	$SP \leftarrow SP-1, (SP) \leftarrow PC_H$ $SP \leftarrow SP-1, (SP) \leftarrow PC_L$ $PC_H \leftarrow 0, PC_L \leftarrow p$	-	3/11	

5.2.2.5. Instrucțiuni de lucru cu stiva și de comandă a UCP

Deși instrucțiunile acestui grup au fost introduse de Zilog în alte categorii [44, 45], pentru o mai bună raportare la criteriul funcționalității, stabilit în §4.3.3, s-a preferat ordonarea din tab.5.7, care evidențiază o atenție sporită acordată lucrului cu stiva.

Tab.5.7.

Mnemonica Z80	Mnemonica 8080	Descrierea operației	Indicatorii afecțați	Nr. CM/ nr. stări	Comentarii
Lucru cu stiva					
PUSH <i>qq</i>	PUSH <i>rp</i> PUSH PSW	$SP \leftarrow SP-1, (SP) \leftarrow qq_H$ $SP \leftarrow SP-1, (SP) \leftarrow qq_L$	-	3/11	<div>Cod <i>qq</i></div> <div>00 BC</div> <div>01 DE</div> <div>10 HL</div> <div>11 AF</div>
PUSH IX		$SP \leftarrow SP-1, (SP) \leftarrow IX_H$ $SP \leftarrow SP-1, (SP) \leftarrow IX_L$	-	4/15	
PUSH IY		$SP \leftarrow SP-1, (SP) \leftarrow IY_H$ $SP \leftarrow SP-1, (SP) \leftarrow IY_L$	-	4/15	
POP <i>qq</i>	POP <i>rp</i> POP PSW	$qq_L \leftarrow (SP)$ $SP \leftarrow SP+1, qq_H \leftarrow (SP)$ $SP \leftarrow SP+1,$	POP AF afec- tează toți indi- catorii de con- diție.	3/10	
POP IX		$IX_L \leftarrow (SP)$ $SP \leftarrow SP+1, IX_H \leftarrow (SP)$ $SP \leftarrow SP+1,$	-	4/14	
POP IY		$IY_L \leftarrow (SP)$ $SP \leftarrow SP+1, IY_H \leftarrow (SP)$ $SP \leftarrow SP+1,$	-	4/14	
EX (SP),HL	XTHL	$H \leftrightarrow (SP+1)$ $L \leftrightarrow (SP)$	-	5/19	

Tab.5.7 (cont.)

EX (SP),IX		$IX_H \leftrightarrow (SP+1)$ $IX_L \leftrightarrow (SP)$	-	6/23	
EX (SP),IY		$IY_H \leftrightarrow (SP+1)$ $IY_L \leftrightarrow (SP)$	-	6/23	
LD SP,HL	SPHL	$SP \leftarrow HL$	-	1/6	
LD SP,IX		$SP \leftarrow IX$	-	2/10	
LD SP,IY		$SP \leftarrow IY$	-	2/10	
Comandă UCP					
NOP	NOP	$PC \leftarrow PC+1$	-	1/4	
HALT	HLT	Oprire UCP	-	1/4	
DI	DI	$IFF_{1,2} \leftarrow 0$	-	1/4	
EI	EI	$IFF_{1,2} \leftarrow 1$	-	1/4	
IM 0		Setare mod 0	-	2/8	Stabilire mod de întreruperi mascabile.
IM 1		Setare mod 1	-	2/8	
IM 2		Setare mod 2	-	2/8	

5.2.2.6. Instrucțiuni de intrare-ieșire

Spre deosebire de instrucțiunile I/E de la procesoarele Intel, puține la număr (2) și relativ simple (numai prin acumulator), Z80 posedă un grup mult mai numeros (24) și puternic (tab.5.8). Astfel, noile instrucțiuni (22) pot să efectueze transferul I/E nu doar prin intermediul acumulatorului, ci prin orice registru de date. Mai mult, cu o singură instrucțiune se poate efectua transferul direct în/din memorie, al unui octet sau al unui bloc de octeți. În mod implicit, registrul C specifică adresa portului I/E, HL conține adresa de memorie destinație/sursă, iar registrul B contorizează numărul de octeți rămași de transferat. Aceste facilități sunt deosebit de avantajoase pentru implementarea eficientă a sistemelor de control.

Tab.5.8.

Mnemonica Z80	Mnemonica 8080	Descrierea operației	Indicatorii afectați	Nr. CM/ nr. stări	Comentarii
IN A,(n)	IN n	$A \leftarrow (n)$	-	3/11	$n \rightarrow A_7 \div A_0$ $A \rightarrow A_{15} \div A_8$
IN r,(C)		$r \leftarrow (c)$	S,Z,H,P/V, N=0	3/12	$C \rightarrow A_7 \div A_0$ $B \rightarrow A_{15} \div A_8$
INI (INput and Increment)		$(HL) \leftarrow (C)$ $B \leftarrow B-1$ $HL \leftarrow HL+1$	N=1; Dacă B=0 după decrementare, atunci Z=1;	4/16	$C \rightarrow A_7 \div A_0$ $B \rightarrow A_{15} \div A_8$
INIR (INput, Increment and Repeat)		$(HL) \leftarrow (C)$ $B \leftarrow B-1$ $HL \leftarrow HL+1$ } se repetă până când B=0.	N=Z=1	4/16 5/21	$C \rightarrow A_7 \div A_0$ $B \rightarrow A_{15} \div A_8$
IND (INput and Decrement)		$(HL) \leftarrow (C)$ $B \leftarrow B-1$ $HL \leftarrow HL-1$	La fel ca INI		
INDR (INput, Decrement and Repeat)		$(HL) \leftarrow (C)$ $B \leftarrow B-1$ $HL \leftarrow HL-1$ } se repetă până când B=0	La fel ca INIR		
OUT (n),A	OUT n	$(n) \leftarrow A$	-	3/11	$n \rightarrow A_7 \div A_0$ $A \rightarrow A_{15} \div A_8$

Tab.5.8 (cont.)

OUT (C),r		(C)←r	-	3/12	C → A _{7÷A₀} B → A _{15÷A₈}
OUTI (OUT and Increment)		(C)←(HL) B←B-1 HL←HL+1	La fel ca INI		
OTIR (OuT, Increment and Repeat)		(C)←(HL) B←B-1 HL←HL+1	se repetă până când B=0		
OUTD (OUT and Decrement)		(C)←(HL) B←B-1 HL←HL-1	La fel ca INI		
OTDR (OuT, Decrement and Repeat)		(C)←(HL) B←B-1 HL←HL-1	se repetă până când B=0		

5.2.2.7. Codurile instrucțiunilor microprocesorului Z80

În Anexa II sunt prezentate codurile mașină ale instrucțiunilor, împreună cu mnemonicele aferente. Din primul tabel se poate deduce logica prin care s-au ales cele 696 de coduri: în afara unui set primar de 252 de coduri, care corespund instrucțiunilor cu opcodul de un octet, au fost rezervate 4 *coduri prefix* (CBh, DDh, EDh, FDh) pentru definirea grupurilor de instrucțiuni care au opcodul pe 2 octeți. A rezultat un al doilea set de coduri, la care primul octet de cod este CBh și un al treilea set, la care prefixul este EDh. Codurile DDh și FDh se utilizează ca prefix pentru instrucțiunile care suportă adresarea indexată, cu IX, respectiv IY. Aceste instrucțiuni sunt marcate cu un fond gri în primele două tabele din Anexa II.

Se poate remarca faptul că, prin intermediul celor patru coduri prefix ar putea fi codificate un număr total de 1276 de instrucțiuni (252+4×256). Din acestea sunt utilizate numai 696, restul de 580 fiind coduri redundante, care stau la baza așa numitelor *instrucțiuni "ascunse"* [32, 33]. Spre exemplu, codurile 30h÷37h din al doilea tabel al Anexei II corespund unor instrucțiuni de tipul SLL (Shift Left Logic). Producătorul nu garantează o execuție corectă a acestor instrucțiuni, datorită unor imperfecțiuni de concepție/fabricație; ca atare nu sunt cuprinse în documentația de firmă.

5.2.3. Tehnici de programare specifice sistemelor cu Z80

În general, toate tehnicile de programare folosite la sistemele cu 8080 rămân valabile și pentru Z80. În plus, utilizarea noilor instrucțiuni conduce la programe mai eficiente decât la sistemele cu microprocesoare Intel. Acestea se referă în special la salvarea și refăcerea registrelor, operațiile pe blocuri de date, ciclurile cu contor sau unele rotiri și deplasări. Câteva din aceste tehnici vor fi exemplificate în continuare.

Macroasamblorul M80 recunoaște și mnemonicele instrucțiunilor microprocesorului Z80, cu condiția ca secvența respectivă de instrucțiuni să fie precedată de o pseudoinstrucțiune .Z80 (cu punct în prima coloană a liniei sursă). Este permisă chiar și mixarea secvențelor de instrucțiuni din cele două seturi de mnemonice, folosind și pseudoinstrucțiunea .8080, ca în exemplul următor:

; Implicit, la începutul unui fișier sursă, asamblorul interpretează mnemonicele 8080	
LXI	H,1000h ; Mnemonice 8080
MVI	M,0Ah
INX	H

.Z80			; Asamblorul este informat că urmează instrucțiuni cu mnemonice Z80
	LD	(HL),0Dh	
	INC	HL	
.8080			; Urmează, din nou, instrucțiuni cu mnemonice 8080.
	LXI	B,10	
	DAD	B	
	MVI	M,0	
;			

Atunci când interpretează mnemonicele instrucțiunilor Z80, macroasamblorul recunoaște aceleași pseudoinstrucțiuni prezentate în §4.3.4.1, cu câteva modificări:

- în locul directivelor DB (Define Byte), DW (Define Word), DS (Define Storage), se pot utiliza și directivele **DEFB**, **DEFW**, respectiv **DEFS**, cu același efect.
- în locul pseudoinstrucțiunii SET de la 8080 se utilizează directiva **DEFL** (deoarece, la Z80, SET este mnemonica unei instrucțiuni).

5.2.3.1. Utilizarea registrelor secundare

Registrele secundare (AF', BC', DE' și HL') reprezintă o soluție avantajoasă pentru schimbările rapide de context care nu reclamă recursivitate. Astfel, salvarea registrelor la intrarea în subrutine și refacerea acestora la ieșire se pot face mai eficient prin instrucțiunile EX AF,AF' și EXX decât prin instrucțiuni de tip PUSH și POP. Mai ales în cazul subrutinelor de tratare a întreruperilor (v. §5.1.2), folosirea registrelor alternative reduce substanțial timpul de răspuns.

Însă lipsa de "identitate" a registrelor secundare limitează folosirea acestora ca simple variabile, prin efortul suplimentar pe care trebuie să-l facă programatorul pentru echilibrarea riguroasă a instrucțiunilor de comutare, pe toate ramurile posibile ale programului. În aceste cazuri este preferabilă varianta cu variabile în stivă.

Schimbările de context reprezintă o cu totul altă problemă față de utilizarea registrelor secundare ca simple variabile. Astfel de situații pot apare frecvent în cazul rutinelor care apelează la rândul lor alte subrutine, ambele folosind aceleași resurse hardware interne.

Spre exemplu, programul principal (princ) apelează rutina subrut, ambele utilizând implicit registrul B drept contor, prin instrucțiuni DJNZ:

princ:	LD	A,0	; Inițializare A.
	LD	B,n	; Repetă ciclul de "n" ori.
ciclu:	CALL	subrut	; A←A+m, dar B=0 după execuția rutinei, deci contorul curent a fost
			; alterat.
	DJNZ	ciclu	; B=FFh după decrementare, deci ciclul devine infinit.
	RET		; Ar trebui să se revină cu A=n×m, dar nu ajunge să se mai execute.
;			
subrut:	LD	B,m	; Repetă bucla de "m" ori.
buc:	INC	A	; A←A+1
	DJNZ	buc	
	RET		; Revine cu A=A+m și B=0.

Evitarea apariției unor astfel de erori impune o schimbare de context la începutul subrutinei subrut, care se poate efectua astfel:

; folosind stiva				; folosind registrele secundare			
subrut:	PUSH	BC	; depune BC în stivă	subrut:	EXX		; BC↔BC' (BC→BC')
	LD	B,m			LD	B,m	
buc:	INC	A		buc:	INC	A	
	DJNZ	buc			DJNZ	buc	
	POP	BC	; extrage BC din stivă		EXX		; BC↔BC' (BC←BC')
	RET				RET		

Partea a II-a - Sisteme cu microprocesoare de 8 biți

Durata schimbării de context este 21T pentru salvarea în stivă, respectiv de numai 8T, pentru utilizarea registrelor secundare. În general, schimbarea de context cu registrele secundare este de 3 până la 8 ori mai rapidă, în funcție de numărul de registre ce trebuie salvate.

Această metodă nu este recomandată atunci când subrutina are nevoie de valorile registrelor din programul apelant sau în cazul a mai mult de două apeluri succesive. Astfel, dacă în exemplul considerat subrută ar chema la rândul ei în buclă o altă subrutină ce ar folosi registrul B, schimbarea cu registrele secundare devine imposibilă. Singura metodă în astfel de situații rămâne utilizarea stivei.

5.2.3.2. Operații pe blocuri de date

Instrucțiunile de transfer și căutare pe bloc simplifică programele de operare cu masive de date, fiind avantajoase în prelucrarea șirurilor de caractere. Pentru determinarea lungimii, precum și pentru operații cum ar fi copierea, concatenarea, sau compararea șirurilor de date, se pot utiliza instrucțiunile cu repetiție: LDIR, LDDR, CPIR și CPDR.

În exemplul care urmează se prezintă o secvență de program care copie un tabel de date de lungime cunoscută, lng (max. 65535), de la adresa sursa la adresa dest.

LD	HL,sursa	; Inițializarea registrelor cu parametrii transferului.
LD	DE,dest	
LD	BC,lng	
LDIR		; Copie șirul "sursa" la adresa "dest".

Dacă se prelucrează șiruri terminate cu un octet nul (00h) sau cu alt caracter special, atunci se vor folosi instrucțiunile LDI, LDD, CPI sau CPD, deoarece permit luarea de decizii intermediare.

Spre exemplu, subrutina de comparare a două șiruri, terminate cu 00h, ale căror adrese de început au fost încărcate în prealabil în registrele HL și DE de către programul apelant, furnizează rezultatul comparării în fanionul Z.

cmpsir:	LD	A,(DE)	; Încarcă în A un octet din șirul adresat cu DE.
	INC	DE	; Pregătește adresa octetului următor în DE.
	OR	A	; Testează dacă este sfârșit de șir.
	RET	Z	; Dacă da, atunci șirurile sunt egale și se revine cu Z=1.
	CPI		; Dacă nu, îl compară cu octetul curent din șirul adresat de HL.
	JR	Z,cmpsir	; Continuă compararea dacă octeții sunt identici.
	RET		; Revenire cu Z=0 dacă octeții diferă (șirurile sunt diferite).

Într-un mod asemănător se pot construi rutine de transfer de blocuri de date, de maximum 256 de octeți, între dispozitive I/E și memorie, folosind instrucțiunile INI, INIR, IND, INDR, respectiv OUTI, OTIR, OUTD, OTDR.

5.2.3.3. Utilizarea registrelor Index

Existența registrelor IX și IY permite reducerea numărului de salvări/refaceri repetate ale registrului HL, atunci când se prelucrează în paralel mai mult de două zone de date.

De exemplu, la adunarea a două șiruri de octeți adresate de registrele HL, respectiv DE, de lungime dată de B, rezultatul este un șir de aceeași lungime, care se depune în memorie folosind registrul IX.

	LD	HL,tabel1	; HL adresează "tabel1".
	LD	DE,tabel2	; DE adresează "tabel2".
	LD	IX,tabel3	; Rezultatul se va depune în "tabel3".
	LD	B,lng	; Încarcă în B lungimea șirului rezultat.
adun:	LD	A,(DE)	; Încarcă în A un element din "tabel2".
	ADD	A,(HL)	; Adună elementul corespunzător din "tabel1".

LD	(IX+0),A	; Depune rezultatul în "tabel3".
INC	HL	; Actualizează registrele de adresă.
INC	DE	
INC	IX	
DJNZ	adun	; Continuă, atât timp cât B≠0.

Adresarea indexată are utilizări multiple, atât pentru referirea la date amplasate la adrese succesive de memorie sau care au o distanță fixă între ele, cât și la adresarea componentelor unor structuri de date.

În exemplul care urmează se prezintă adunarea a două elemente de 16 biți (elem), care fac parte din două structuri de date de același tip, adresate cu IX și IY. Rezultatul se va depune în structura adresată de IX, într-un element rezervat special în acest scop (suma).

```
; struct1->suma=struct1->elem+struct2->elem
```

LD	IX,struct1	; Încarcă în IX adresa structurii struct1.
LD	IY,struct2	; Încarcă în IY adresa structurii struct2.
LD	A,(IX+elem)	; elem este deplasamentul la care se află operandul.
ADD	A,(IY+elem)	; Adună LSB al celor două elemente.
LD	(IX+suma),A	; Memorează LSB al rezultatului.
LD	A,(IX+elem+1)	
ADC	A,(IY+elem+1)	; Adună MSB al celor două elemente.
LD	(IX+suma+1),A	; Memorează MSB al rezultatului.

5.2.3.4. Utilizarea deplasărilor în operațiile aritmetice

Instrucțiunile de *deplasare aritmetică*, precum și instrucțiunile de adunare/scădere pe 16 biți se folosesc în subrutinele aritmetice cu întregi de 32 de biți și cu numere reale în virgulă mobilă. Reducerea lungimii acestor subrutine se obține și prin realizarea deplasărilor direct în memorie, fără aducerea operanzilor în registre.

Spre exemplu, prin combinarea instrucțiunilor de rotire și deplasare la stânga, RL și respectiv SLA, înmulțirea cu 4 a unei variabile de 2 octeți, aflată în memorie, decurge astfel:

LD	HL,var	; Încarcă în HL adresa variabilei (adresa LSB).
CALL	mul4	; Apelează rutina de înmulțire cu 4.
;		
mul4:		
SLA	(HL)	; LSB×2
INC	HL	; HL= adresa MSB
RL	(HL)	; MSB×2
DEC	HL	; Revine la LSB.
SLA	(HL)	; LSB×2
INC	HL	; Revine la MSB.
RL	(HL)	; MSB×2
RET		
;		
var:	DEFS 2	; Rezervă doi octeți pentru variabilă (în RAM).

În mod similar, prin utilizarea instrucțiunilor SRA și RR se poate implementa împărțirea numerelor întregi.

Instrucțiunile pentru *deplasare zecimală*, RLD și RRD, își găsesc utilizarea în aritmetica binar-zecimală (BCD). Astfel, deplasarea la stânga cu o poziție echivalează cu o înmulțire cu 10, iar la dreapta cu o împărțire prin 10. Ambele operații sunt necesare la înmulțirea și împărțirea de numere BCD, precum și pentru reducerea numerelor zecimale neîntregi la numere întregi.

Exemplu: înmulțirea cu 10 a unui număr în BCD, fără semn, aflat în memorie la adresa din HL și având lungimea dată în B.

mul10:	XOR	A	; A ← 0
dec:	RLD		; Deplasare spre stânga, cu o poziție.
	INC	HL	; Actualizează adresa.
	DJNZ	dec	; Deplasează toți octeții.

5.3. Introducerea timpului ca variabilă în sistemele cu Z80

Utilizarea de circuite programabile specializate pentru materializarea taskurilor de timp este soluția cea mai folosită și în cazul sistemelor cu Z80. Facilitățile și larga răspândire a dispozitivului Intel 8253 (v. §4.5.1) au condus la o folosire a acestuia și în sistemele cu Z80, cu mici adaptări HW specifice (v. §5.1.5.1). Însă, o astfel de soluție prezintă dezavantajul unei rezolvări mai complicate a tratării întreruperilor, deoarece nu se poate utiliza modul 2, cel mai puternic la Z80.

Folosirea circuitului dedicat CTC-Z80, cu variante direct compatibile cu Z80, Z80A și Z80B, este soluția optimă în acest caz.

5.3.1. Circuitul CTC-Z80 (Counter/Timer Circuit)

Acest dispozitiv este un numărător/temporizator programabil, cu patru canale independente, așa cum se arată în schema bloc internă din fig. 5.25. Funcționarea CTC-Z80 este sincronizată prin intermediul semnalului CLK, care se conectează la semnalul de tact al UCP (Φ).

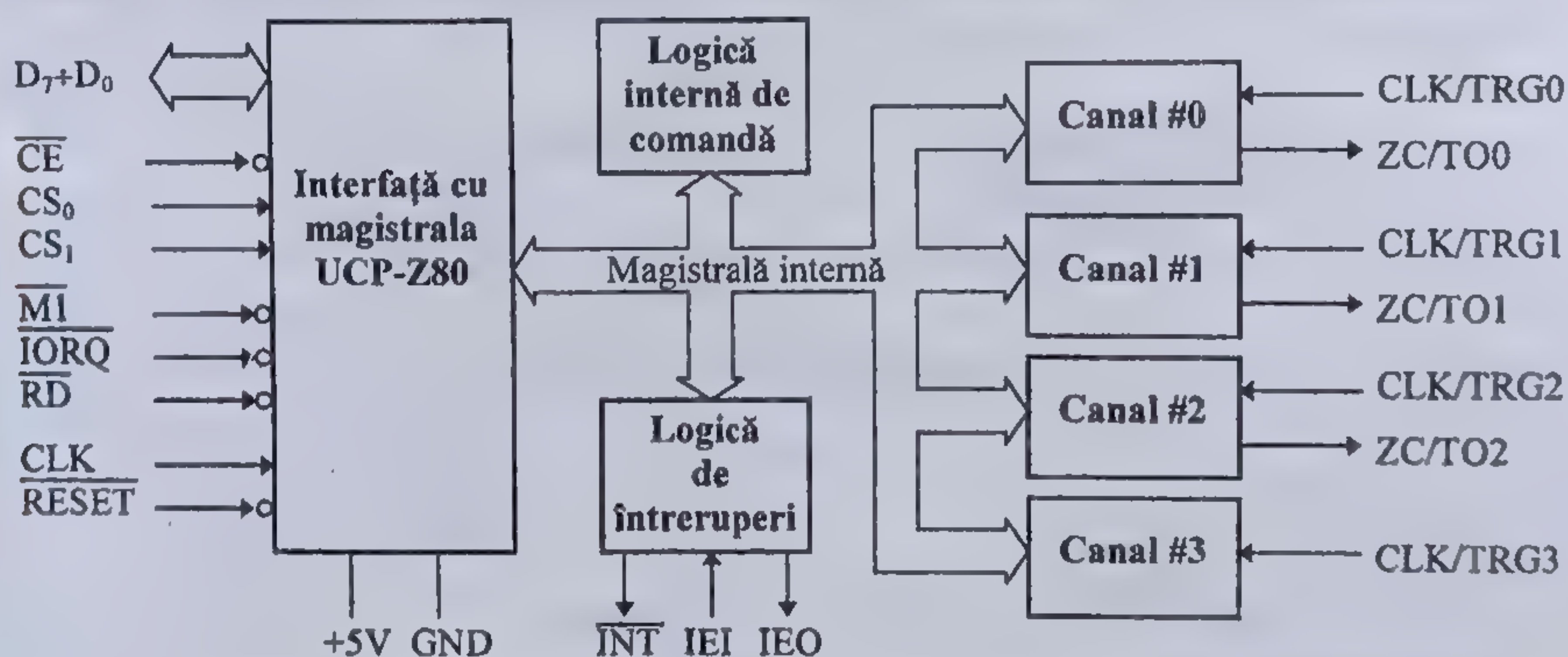


Fig. 5.25. Structura internă a circuitului Z80-CTC

Tab. 5.9.

\overline{CE}	\overline{IORQ}	$\overline{M1}$	\overline{RD}	CS_1	CS_0	Acțiunea
0	0	1	0	0	0	Citire numărător canal #0
0	0	1	1	0	0	Înscriere numărător canal #0
0	0	1	0	0	1	Citire numărător canal #1
0	0	1	1	0	1	Înscriere numărător canal #1
0	0	1	0	1	0	Citire numărător canal #2
0	0	1	1	1	0	Înscriere numărător canal #2
0	0	1	0	1	1	Citire numărător canal #3
0	0	1	1	1	1	Înscriere numărător canal #3
x	0	0	1	x	x	Identificare solicitant întrerupere

Interfața cu magistrala Z80 asigură transferul bidirecțional de date (D_7+D_0), selecția (\overline{CE} , CS_0 , CS_1) și controlul operațiilor de citire/înscriere a CTC-Z80 ($\overline{M1}$, \overline{RD} , \overline{IORQ}). Logica

internă de comandă asigură controlul funcționării circuitului, în corespondență cu regimurile impuse de UCP, iar logica de întreruperi permite conectarea într-un lanț de priorități și funcționarea în modul 2 de întreruperi. Canalele pot fi selectate pe liniile CS_0 și CS_1 (Channel Select), care se conectează de obicei la liniile A_0 și respectiv A_1 ale magistralei de adrese. Acțiunea combinată a semnalelor de comandă are efectele prezentate în tab.5.9.

Deoarece circuitul nu posedă linie \overline{WR} , rolul acesteia este preluat de linia \overline{RD} , printr-o complementare internă. Prin intermediul liniilor CLK/TRG (Clock/Trigger), toate canalele pot fi comandate de evenimente externe, iar primele trei canale pot semnaliza încheierea unei proceduri prin semnalele ZC/TO (Zero Count/Timeout).

După conectarea tensiunii de alimentare, CTC-Z80 rămâne într-o stare nedefinită până la activarea semnalului \overline{RESET} , care aduce circuitul în starea inițială. În această stare, circuitul așteaptă comenzile de programare din partea UCP, fără de care nu poate funcționa.

Activarea semnalului \overline{RESET} în timpul funcționării circuitului determină oprirea canalelor, invalidarea întreruperilor, dezactivarea ieșirilor ZC/TO și \overline{INT} , comandă IEO=IEI și trece liniile $D_7 \div D_0$ în starea de înaltă impedanță.

5.3.1.1. Modurile de lucru ale circuitului CTC-Z80

Structura internă a canalelor este identică și se prezintă în schema bloc din fig.5.26. Fiecare canal poate lucra în unul din cele două moduri de bază: *numărător* (counter) sau *temporizator* (timer).

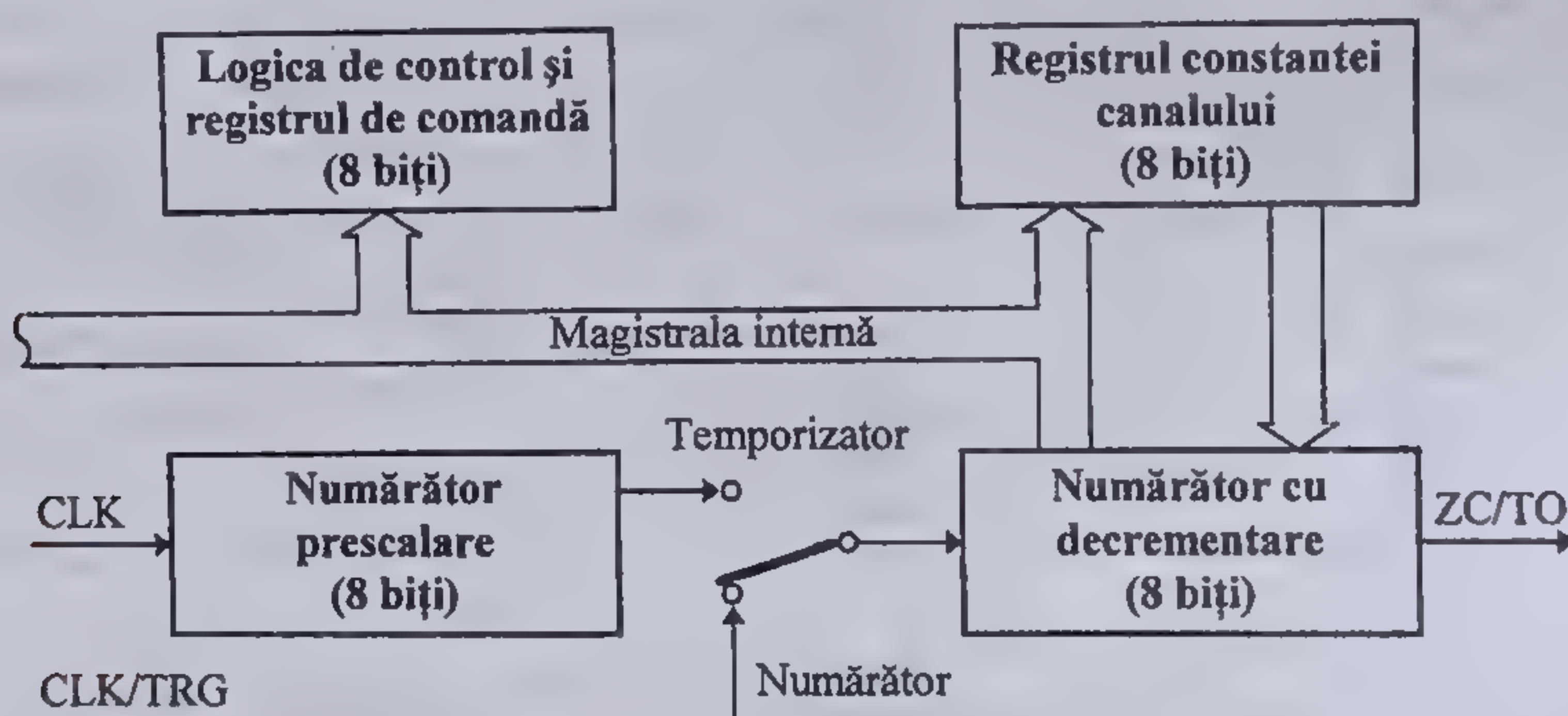


Fig.5.26. Structura internă a unui canal de numărare/temporizare

În modul *numărător*, un canal contorizează fronturile crescătoare sau descrescătoare (selectabil software) ale unui semnal extern, aplicat la intrarea CLK/TRG. După detectarea numărului de evenimente programat, canalul emite un impuls de cca. $1,5T_{CLK}$ pe ieșirea ZC/TO și generează o cerere de întrerupere, în cazul în care a fost programat să lucreze cu întreruperile. În mod automat, funcționarea este reluată sistematic, în același mod și cu aceeași constantă, până la o nouă programare.

În modul *temporizator*, circuitul numără perioadele de tact intern, multiplicat prin 16 sau 256 de către un divizor de frecvență (prescaler). Declanșarea numărării se poate face prin program sau de către semnalul extern CLK/TRG. La atingerea numărului de impulsuri programat, canalul reacționează la fel ca în modul numărător. Astfel, în acest mod pot fi generate, pe ieșirea ZC/TO, impulsuri de durată programabilă (nivelul logic "1") sau pot fi măsurate intervale de timp.

Relația de calcul al intervalului de timp este:

$$T_{ZC/TO} = T_{CLK} \times P \times N,$$

În care T_{CLK} este perioada semnalului de tact al sistemului, P e valoarea de prescalare (16 sau 256), iar N este valoarea încărcată în registrul constantei de timp a canalului, între 1 și 256. Rezultă că intervalul minim de timp ce poate fi măsurat în acest mod este $16T_{CLK}$, iar cel maxim $65536T_{CLK}$.

Pentru materializarea modurilor de funcționare descrise mai sus, fiecare canal este prevăzut cu o logică proprie de comandă, două numărătoare de 8 biți și două registre pentru stabilirea modului de operare, respectiv pentru constanta de timp (fig.5.26). Numărătorul cu decrementare (down counter) poate fi înscris numai cu valoarea programată în registrul constantei canalului, după programare și ori de câte ori se atinge valoarea 0, prin decrementare. Microprocesorul poate numai citi conținutul numărătorului decrementor. În funcție de modul de lucru, contorizare sau temporizare, acest numărător primește semnalul de tact din exterior sau de la numărătorul de prescalare, care asigură divizarea acestuia prin 16 sau 256.

5.3.1.2. Programarea CTC-Z80

Registrul de comandă al fiecărui canal poate fi programat de către UCP cu un cuvânt de comandă de 8 biți, CW, transmis pe liniile D_7-D_0 .

D_7	D_6	D_5^*	D_4	D_3^*	D_2	D_1	D_0
Validare întreruperi	Mod de lucru	Factor prescalare	Front activ CLK/TRG	Mod de declanșare	Urmează constanta	Inițializare	1
1 - validare 0 - invalidare	1 - numărător 0 - temporizator	1 - 256 0 - 16	1 - crescător 0 - scăzător	1 - CLK/TRG	1 - Da 0 - Nu	1 - Da 0 - Nu	CW

În continuare se vor face unele precizări referitoare la biții cuvântului, care nu rezultă direct din numele și valorile posibile ale acestora. Prin intermediul bitului D_7 se poate activa sau dezactiva generarea unei întreruperi la încheierea unui ciclu de numărare/temporizare. Dacă întreruperile sunt invalidate în timpul funcționării, atunci este necesar să se țină cont de asincronismul dintre transmiterea comenzii de către UCP și posibilitatea acceptării întreruperii de la canalul deja activat. Sincronizarea CTC cu UCP se poate realiza prin următoarea secvență de instrucțiuni:

LD	A, 01h	; Încarcă în acumulator cuvântul de comandă cu $D_7=0$.
DI		; Dezactivează acceptarea întreruperilor la UCP.
OUT	(CTC+i), A	; Comandă invalidarea generării întreruperilor pe canalul i.
EI		; Activează întreruperile la UCP.

Dacă se dorește ca, la sfârșitul unui ciclu de numărare/temporizare, canalul să lanseze o cerere de întrerupere, pe lângă faptul că D_7 trebuie să fie 1, este necesar ca dispozitivului CTC-Z80 să-i fie transmis la programare și vectorul de întrerupere.

Programarea canalului în modul temporizator ($D_6=0$) necesită și specificarea biților D_5 (factorul de prescalare) și D_3 (modul de declanșare). Dacă $D_3=0$, funcționarea este declanșată după înscrierea constantei de timp, pe următorul front pozitiv al semnalului CLK. Pentru $D_3=1$, declanșarea se realizează prin semnalul extern CLK/TRG, cu frontul activ stabilit prin bitul D_4 . Decrementarea numărătorului se face cu semnalul primit de la numărătorul divizor (prescaler).

În modul numărător ($D_6=1$), valoarea biților D_5 și D_3 nu mai are importanță (*). Decrementarea numărătorului este determinată numai de semnalul extern CLK/TRG (pe frontul activ selectat cu bitul D_4), sincron cu semnalul de tact CLK. Pentru o funcționare corectă în acest mod este necesar ca semnalul CLK/TRG să aibă o perioadă de cel puțin două ori mai mare decât cea a semnalului CLK.

Prin bitul D_2 se specifică faptul că, după cuvântul de comandă, UCP transmite canalului constanta de timp. Acest lucru este obligatoriu la începerea funcționării unui canal. Ulterior, dacă constanta nu trebuie modificată, orice cuvânt de comandă va conține $D_2=0$. Dacă, în timpul funcționării canalului se transmite o nouă constantă, după un cuvânt de comandă corespunzător, aceasta va fi luată în considerație abia după încheierea ciclului curent de numărare, odată cu prima trecere a numărătorului prin zero. Dacă este necesară stoparea funcționării unui canal fără a fi afectați biții registrului de comandă, se înscrie un cuvânt de comandă având bitul $D_1=1$ (software reset). În cazul în care $D_2=D_1=1$, canalul poate fi inițializat cu o nouă constantă de timp și își va relua funcționarea imediat după încărcarea acesteia.

Toate cele arătate mai sus permit programatorului să stabilească structura cuvântului de comandă, în corespondență cu cerințele unei aplicații la un moment dat.

Registrul constantei de timp poate fi înscris cu orice valoare cuprinsă între 0 și 255. Valoarea 0 este echivalentă cu 256. După cum s-a arătat mai sus, încărcarea constantei succede unui cuvânt de comandă cu $D_2=1$.

Diagramele de timp la citire și scriere, precum și la funcționarea în cele două moduri de operare pot fi găsite în [44, 45].

Programarea CTC-Z80 se face prin transmiterea cuvintelor de comandă și a constantei de timp pentru fiecare canal utilizat. Dacă cel puțin unul din cele patru canale lucrează cu întreruperile, atunci este necesară și transmiterea vectorului de întrerupere. Acesta este înscris numai pentru canalul #0, deoarece logica internă de întreruperi (v.fig.5.25) modifică automat vectorul plasat pe magistrală de către CTC, în conformitate cu canalul solicitant. În fig.5.27 este prezentată sintetic secvența de programare a circuitului CTC-Z80.

Vectorul de întrerupere (VI) este de un octet și are bitul cel mai puțin semnificativ egal cu 0, specific modului 2 de întreruperi al microprocesorului Z80. Practic, programatorul trebuie să precizeze numai primii 5 biți mai semnificativi (D_7+D_3), deoarece, așa cum s-a arătat, CTC stabilește automat valoarea biților D_2 și D_1 în funcție de canalul care solicită întreruperea.

Logica de întreruperi asigură și rezolvarea internă a priorităților în tratarea cererilor suprapuse pe cele 4 canale, prin acordarea priorității maxime canalului 0.

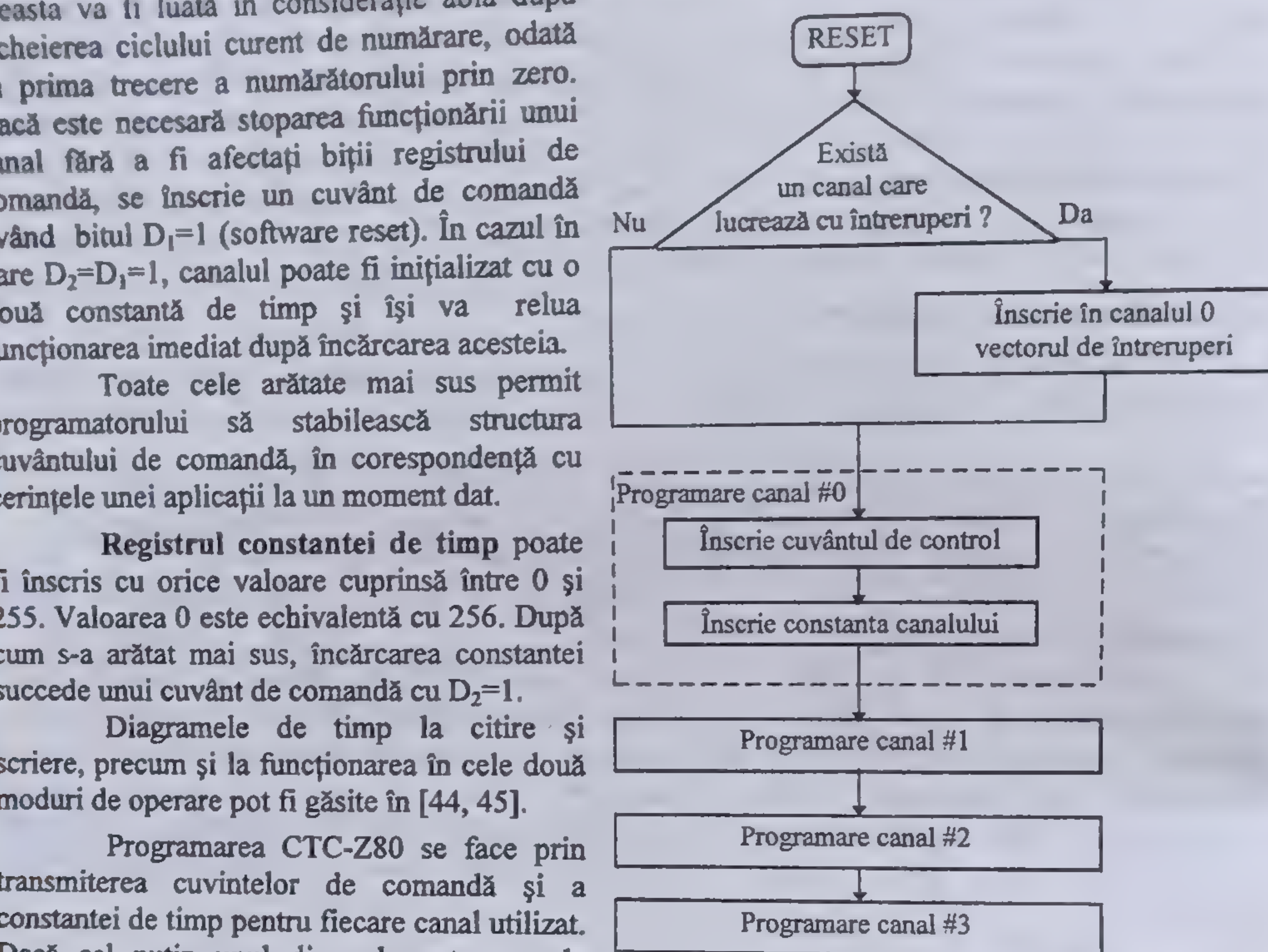


Fig.5.27. Programarea circuitului CTC-Z80

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	x	x	0
Precizați de utilizator					Introduși de CTC		
					0 0 - canalul #0		
					0 1 - canalul #1		
					1 0 - canalul #2		
					1 1 - canalul #3		

VI

Partea a II-a - Sisteme cu microprocesoare de 8 biți

La cele arătate în fig.5.27 trebuie adăugat și faptul că, în prealabil, este necesară inițializarea registrului I al microprocesorului Z80 cu octetul superior al adresei tabelului de întreruperi.

Exemplu: Să se programeze canalele #1 și #2 ale unui CTC-Z80, selectat cu linia A₆, să funcționeze în modurile numărător, respectiv temporizator, cu întreruperi. La adresa 2010h se află tabelul de întreruperi, care conține adresele rutinelor de tratare pentru cele 4 canale CTC.

LD	A,20h	; High(2010h).
LD	I,A	; Inițializare registru I al microprocesorului Z80.
LD	A,10h	; Low(2010h).
OUT	(40h),A	; Înscrie vectorul de întrerupere pe adresa canalului #0 al CTC.
LD	A,11000111b	; Numărător pe front căzător (CLK/TRG), urmează constanta, inițializare, generare întrerupere la terminarea numărării.
OUT	(41h),A	; Înscrie cuvântul de comandă pe adresa canalului #1 al CTC.
LD	A,(const1)	; Înscrie constanta în canalul #1 al CTC.
OUT	(41h),A	; Temporizator, ×256, declanșat pe front căzător (CLK/TRG), urmează constanta, inițializare, generare întrerupere la terminarea temporizării.
LD	A,10101111B	; Înscrie cuvântul de comandă pe adresa canalului #2 al CTC.
OUT	(42h),A	; Înscrie constanta în canalul #2 al CTC.
LD	A,(const2)	; Programare mod 2 de întreruperi mascabile.
OUT	(42h),A	; Validare întreruperi la Z80.
IM	2	
EI		
.....		
tabint:	ORG 2010h	; Inițializare statică a tabelului de întreruperi (amplasat în ROM).
	DW rut#0	; rut#0 = adresa rutinei de tratare a întreruperii solicitate de canalul #0
	DW rut#1	; rut#1 - similar, pentru canalul #1
	DW rut#2	; rut#2
	DW rut#3	; rut#3

Inițial se încarcă registrul I al microprocesorului cu octetul superior al adresei tabelului de întreruperi, tabint. Apoi, se transmite vectorul de întrerupere pentru canalul #0, reprezentând octetul inferior al adresei tabelului. Pentru funcționarea canalului #1 ca numărător, se transmite cuvântul de comandă, în care se specifică funcționarea cu întreruperi ($D_7=1$), modul numărător ($D_6=1$), frontul activ descrescător al semnalului CLK/TRG ($D_4=1$) și inițializare canal ($D_2=D_1=1$) cu valoarea constantei simbolice const1. Se programează apoi canalul #2 în modul temporizator, cu factorul de prescalare 256 ($D_5=1$), declanșare pe frontul căzător ($D_4=0$) al semnalului CLK/TRG ($D_3=1$) și inițializare cu valoarea constantei simbolice const2.

Întrucât CTC-Z80 funcționează în modul 2 de întreruperi, înainte de activarea mecanismului de întreruperi mascabile (EI), trebuie să se execute instrucțiunea IM 2.

5.3.2. Organizarea sistemelor de introducere a timpului (SIT) cu CTC-Z80

Organizarea SIT cu CTC-Z80 implică aceleași faze ca și în cazul utilizării circuitelor 8253 (v. §4.5.2) și anume: stabilirea numărului de dispozitive CTC în funcție de numărul taskurilor de timp, conectarea acestora la magistralele sistemului și programarea fiecărui canal de timp, în corespondență cu tipul taskului ce trebuie implementat. De asemenea, dacă se lucrează cu întreruperile, trebuie organizat tabelul cu adresele rutinelor de servire a întreruperilor.

Exemplu: Să se realizeze un SIT pentru 7 taskuri de timp, folosind 2 circuite CTC-Z80 (CTC-A și CTC-B). Cel puțin unul din canalele fiecărui CTC lucrează în regim de întreruperi. Tabelul cu adresele subrutinelor de servire pentru SIT se amplasează în memorie începând de la

adresa 8000h. Adresele celor două dispozitive sunt 40h și respectiv 50h, CTC-A fiind prioritar la solicitarea de întreruperi față de CTC-B. În fig.5.28 este prezentată conectarea circuitelor CTC la magistralele sistemului cu Z80.

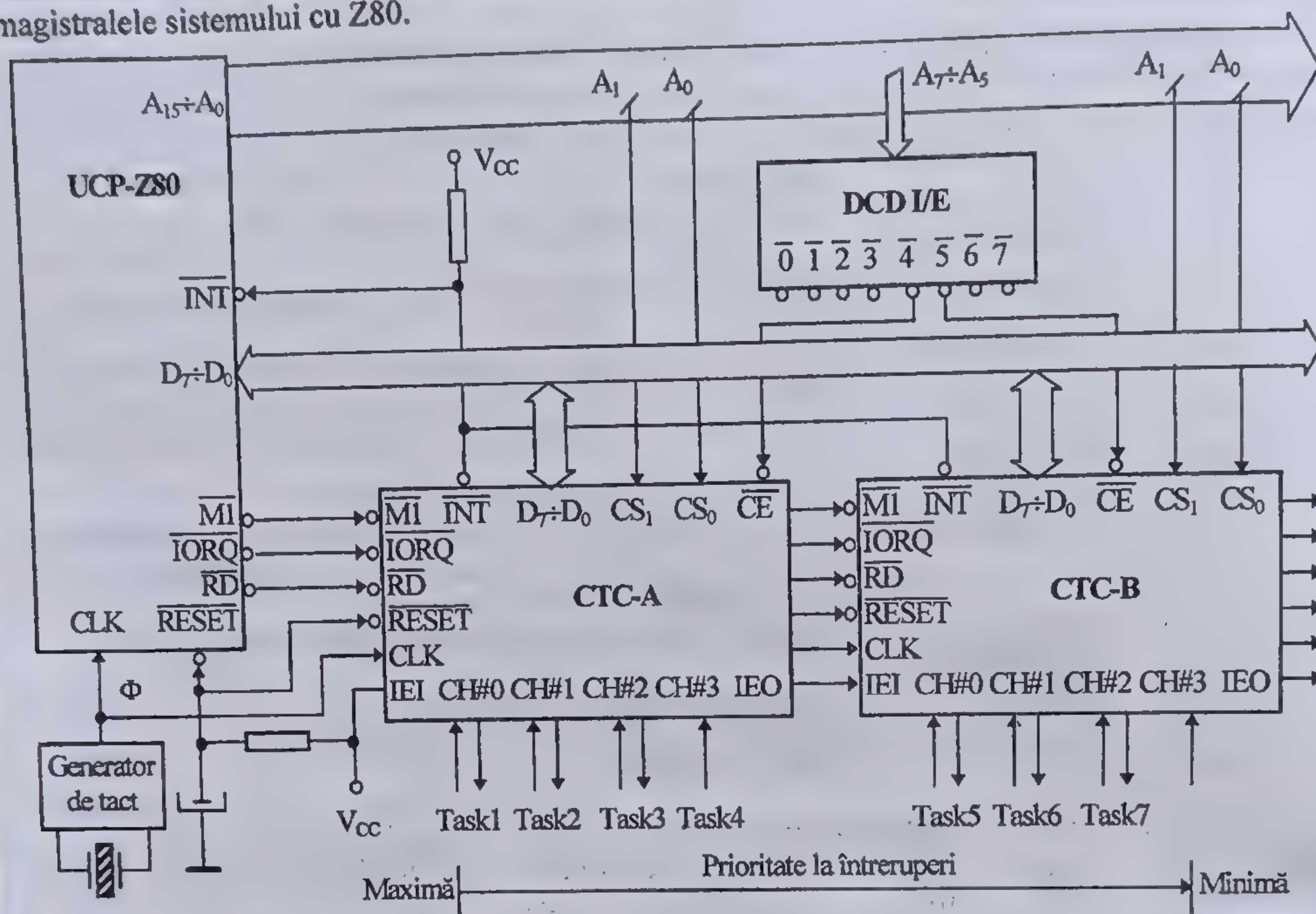


Fig.5.28. Schema unui SIT cu CTC-Z80 pentru 7 task-uri de timp

Programarea dispozitivelor CTC din SIT se realizează astfel:

```

CTC_A EQU 40h
CTC_B EQU 50h
; Programare Z80 în modul 2 de întreruperi.
IM 2
LD A,HIGH(tabint)
LD I,A ; I = octetul superior al adresei tabelului de întreruperi.
; Inițializare tabel de întreruperi (amplasat în RAM).
LD HL,itask1 ; HL = adresa rutinel de tratare a întreruperii de la CTC-A, #0.
LD (tabint),HL ; Memorează adresa în tabelul de întreruperi.
LD HL,itask2 ; Similar, pentru celelalte taskuri.
LD (tabint+2),HL
LD HL,itask3
LD (tabint+4),HL
LD HL,itask4
LD (tabint+6),HL
LD HL,itask5
LD (tabint+8),HL
LD HL,itask6
LD (tabint+10),HL
LD HL,itask7
LD (tabint+12),HL

```


; Programare CTC-A

```
LD    A,LOW(tabint)
OUT   (CTC_A),A      ; Înscrie vectorul de întreruperi, pe adresa canalului #0 al CTC-A.
LD    A,(CWTSK1)
OUT   (CTC_A),A      ; Înscrie cuvântul de comandă pentru canalul #0 al CTC-A.
LD    A, (const1)
OUT   (CTC_A), A     ; Înscrie constanta pentru canalul #0 al CTC-A.
LD    A,(CWTSK2)
OUT   (CTC_A+1),A    ; Similar, se programează și celelalte canale ale CTC-A.
LD    A, (const2)
OUT   (CTC_A+1), A
LD    A,(CWTSK3)
OUT   (CTC_A+2),A
LD    A, (const3)
OUT   (CTC_A+2), A
LD    A,(CWTSK4)
OUT   (CTC_A+3),A
LD    A, (const4)
OUT   (CTC_A+3), A
```

; Programare CTC-B

```
LD    A,LOW(tabint)+8
OUT   (CTC_B),A      ; Înscrie vectorul de întreruperi, pe adresa canalului #0 al CTC-B.
LD    A,(CWTSK5)
OUT   (CTC_B),A      ; Înscrie cuvântul de comandă pentru canalul #0 al CTC-B.
LD    A, (const5)
OUT   (CTC_B), A     ; Înscrie constanta pentru canalul #0 al CTC-B.
LD    A,(CWTSK6)
OUT   (CTC_B+1),A    ; Similar, se programează canalele #1 și #2 ale CTC-B.
LD    A, (const6)
OUT   (CTC_B+1), A
LD    A,(CWTSK7)
OUT   (CTC_B+2),A
LD    A, (const7)
OUT   (CTC_B+2), A
EI                                ; Validare întreruperi la Z80
```

;

DSEG

ORG 8000h

tabint: DS 2*4*2

; Rezervă spațiu de memorie pentru tabelul de întreruperi: 2 circuite
; CTC × 4 canale × 2 octeți (adresa de tratare)

Cuvintele de comandă CWTSK1+CWTSK7 și constantele de timp const1+const7 corespund taskurilor pe care le execută canalele utilizate ale celor două dispozitive CTC. Se poate observa că, nefiind folosit, canalul #3 al CTC-B nu este programat, fiind disponibil în vederea unei eventuale extinderi a SIT.

5.3.3. Sisteme pentru întreruperi vectorizate cu CTC-Z80

După cum s-a văzut în §5.1.2.2, tratarea cererilor multiple de întrerupere la sistemele cu Z80 se poate realiza, prin utilizarea controlerelor 8259 ale familiei Intel, numai în modul 0. Această soluție nu beneficiază de flexibilitatea și timpul mai redus de răspuns al modului 2. Circuitele CTC-Z80 pot fi folosite ca și controlere de întreruperi, cu avantajul folosirii modului 2. Acest lucru este posibil deoarece toate cele 4 canale ale unui CTC pot lansa cereri de întrerupere către UCP la terminarea numărării. Pentru ca lansarea unei întreruperi printr-un canal să fie

corelată cu evenimente externe, este necesar ca acesta să funcționeze ca numărător, cu constanta corelată cu evenimente externe, este necesar ca acesta să funcționeze ca numărător, cu constanta 01h. La apariția unui front activ pe linia CLK/TRG, numărătorul canalului este adus în zero și lansează o cerere de întrerupere către microprocesor. În același timp, constanta 01h este automat reîncărcată în numărătorul cu decrementare, canalul fiind pregătit să detecteze următorul front activ. Întreruperea determinată de acesta nu va putea fi luată în considerare până la încheierea tratării solicitării anterioare, când, în rutina de tratare corespunzătoare, microprocesorul execută instrucțiunea RETI.

Ținând cont de cele arătate mai sus, se pot organiza SINT cu un număr mare de niveluri de întrerupere, în modul 2, însă folosind numai dispozitivele familiei Zilog.

Exemplu: Se consideră exemplul din §5.3.2, în care taskurile vizează tratarea solicitărilor de întrerupere de la 8 dispozitive externe. Utilizând toate canalele disponibile ale celor două circuite CTC-Z80 din fig.5.28 (CTC-A și CTC-B), rezultă un sistem de întreruperi cu 8 niveluri.

Dezavantajul unei astfel de structuri constă în faptul că prioritățile sunt fixe și nu pot fi modificate prin program. Programarea sistemului se face la fel ca în §5.3.2, cu $CWTSK1 \div CWTSK8 = 11010111b$ (cereri active pe front crescător) și $const1 \div const8 = 01h$. Este posibilă mascarea individuală a fiecărei linii de întrerupere, printr-un cuvânt de comandă de invalidare a întreruperilor (01h) inclus într-o secvență de sincronizare cu UCP-Z80, așa cum s-a arătat în §.5.3.1.2.

5.4. Accesul direct la memorie în sistemele cu Z80

Accesul direct la memorie în sistemele cu Z80 se bazează tot pe facilitatea de cedare a magistralelor, cu care este prevăzut și acest microprocesor. Controlul cedării/preluării magistralei de sistem se realizează, ca și la procesoarele Intel, cu circuite specializate. Răspândirea controlerelor DMA 8257/8237 a condus la utilizarea acestora la realizarea accesului direct la memorie și în sistemele cu Z80. Dar, dispozitivul DMA-Z80, direct compatibil cu Z80, ușurează proiectarea și elimină logica suplimentară necesară compatibilizării celor două tipuri de sisteme. În plus, pot fi asigurate unele regimuri de lucru pe care controlerul Intel nu le posedă. Spre exemplu, acest controler poate fi programat să funcționeze în trei moduri de transfer:

a) **octet cu octet** (byte-at-a-time), când se efectuează transferul unui octet, iar între două transferuri magistralele sistemului sunt cedate UCP;

b) **condiționat** (burst), când transferul continuă până ce unul din participanții la transfer dezactivează cererea de acces direct la memorie. Controlerul DMA cedează controlul asupra magistralelor după încheierea ciclului DMA în curs de desfășurare (pentru octetul curent);

c) **continuu** (continuous), când transferul datelor se realizează până la terminarea blocului programat. Dacă unul din participanți anunță, în timpul operării, că este inapt pentru transfer, controlerul DMA așteaptă până când participantul respectiv redevine activ.

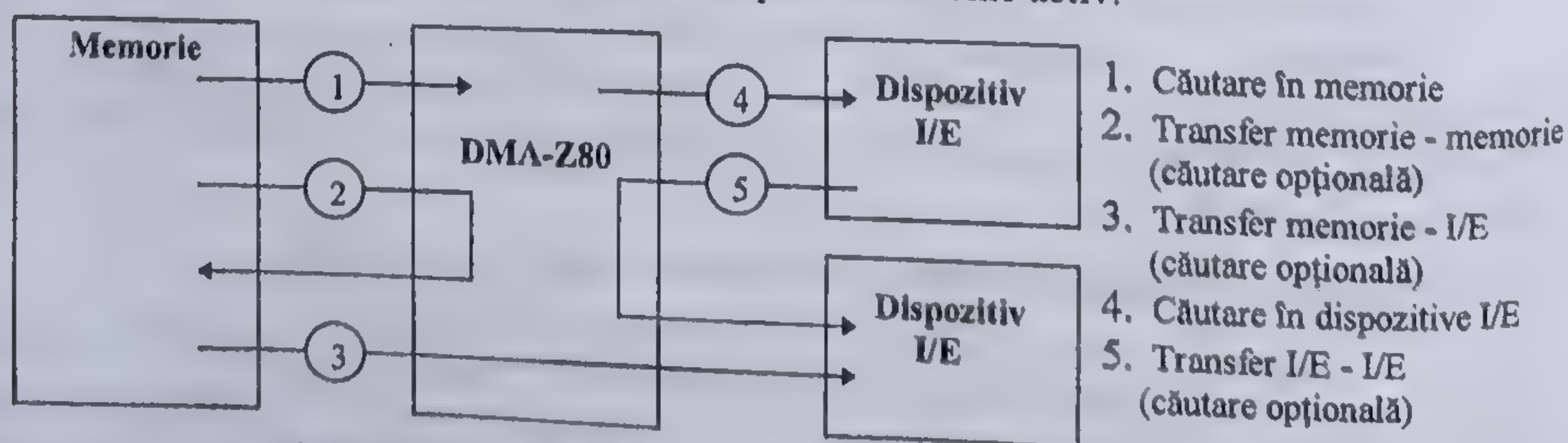


Fig.5.29. Modalitățile de funcționare ale circuitului DMA-Z80

Modurile de funcționare de mai sus se realizează atât pentru *transferuri de date* cât și/sau pentru operații de *căutare de date* (data search). Structura circuitului DMA-Z80 furnizează adrese duale complete (sursă și destinație) prin două porturi de 16 biți. Drept urmare, sunt posibile transferuri nu numai între memorie și dispozitive I/E, ci și între două zone de memorie sau două dispozitive I/E. În fig.5.29 sunt prezentate toate posibilitățile de funcționare ale controlerului DMA-Z80.

În timpul operațiilor de căutare (fără transfer) datele sunt citite dintr-un port sursă și sunt comparate, octet cu octet, cu conținutul unui registru de coincidență (byte match register), programabil. Opțional, octetul din registrul de coincidență poate fi mascat, astfel ca numai anumiți biți să fie comparați cu datele citite. Viteza de căutare ajunge până la $\frac{1}{2}$ din frecvența maximă de tact (1,25 Moct/s la DMA-Z80 și 2 Moct/s la DMA-Z80A).

Transferurile de date sau căutările pot fi configurate să se oprească după efectuarea numărului de cicluri DMA programate sau doar să genereze o întrerupere.

5.4.1. Structura internă a circuitului DMA-Z80

Structura internă a circuitului DMA-Z80 este prezentată în schema bloc din fig.5.30. Acesta are 16 linii de adresă ($A_{15} \div A_0$) și 8 linii de date ($D_7 \div D_0$), cu care se conectează la magistralele de adrese și de date ale sistemului.

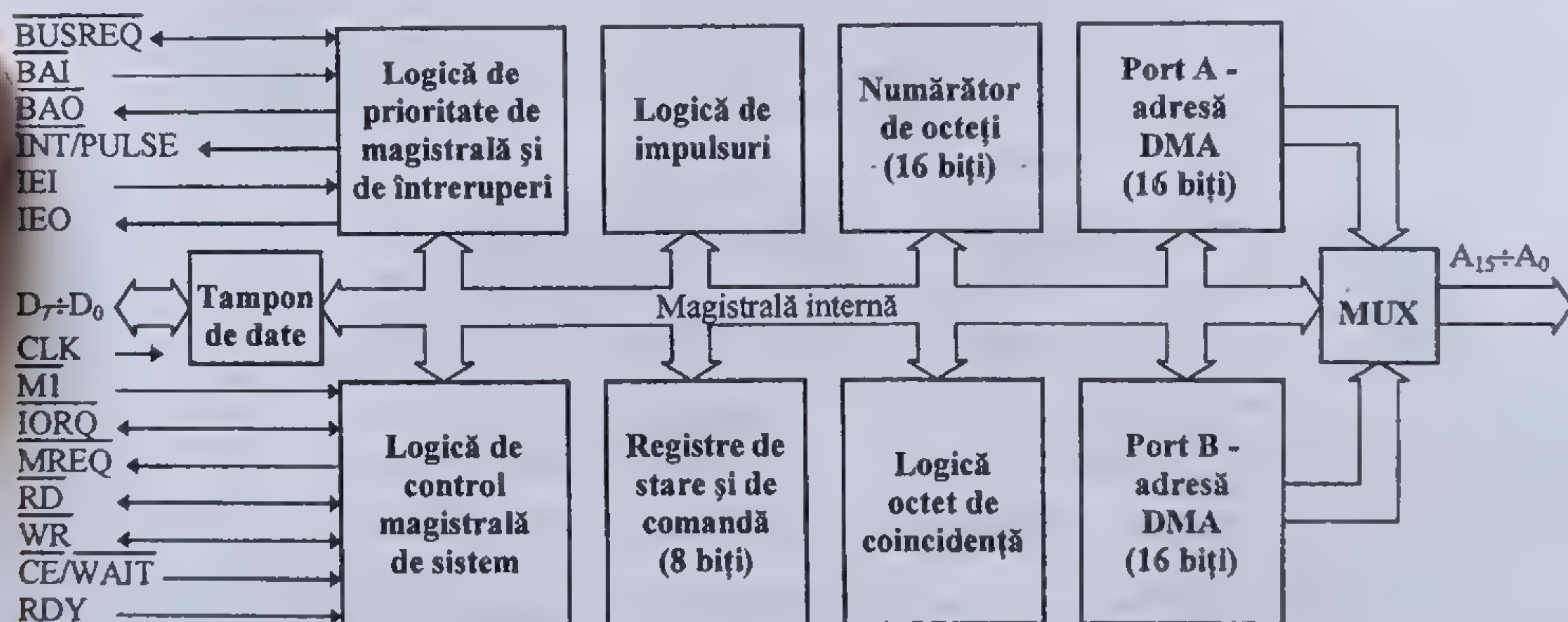


Fig.5.30. Schema bloc internă a circuitului DMA-Z80

De asemenea, posedă linii de comandă direct compatibile cu microprocesorul Z80: \overline{MI} , \overline{IORQ} , \overline{MREQ} , \overline{RD} , \overline{WR} și are facilități specifice unui controler DMA.

În regim de slave, circuitul DMA este controlat de microprocesor ca un dispozitiv I/E (\overline{IORQ} , \overline{RD} , \overline{WR} - intrări), în vederea înscrierii registrelor de comandă sau citirii registrelor de stare. În urma unei cereri pe linia RDY (Ready), circuitul trece în regim de master DMA: preia controlul asupra sistemului (\overline{IORQ} , \overline{MREQ} , \overline{RD} , \overline{WR} - ieșiri) și suspendă activitatea UCP. Oprirea microprocesorului are loc ca urmare a solicitării pe linia BUSREQ și a recepționării confirmării BAI (Bus Acknowledge Input). Dezavantajul existenței unui singur canal DMA pe cip este suplinit de posibilitatea utilizării mai multor circuite DMA-Z80 conectate într-un lanț de priorități, care permite arbitrarea accesului la magistrala sistem, prin intermediul liniilor BAI și BAO (Bus Acknowledge Output), așa cum se arată în fig.5.31.

În plus, circuitele DMA pot fi conectate și în lanțul de priorități de întreruperi, prin liniile IEI și IEO. Întreruperile pot fi programate pentru trei situații:

- 1) Întrerupere la activarea semnalului RDY (înaintea cererii de magistrală pe $\overline{\text{BUSREQ}}$);
- 2) Întrerupere la coincidență (la căutare);
- 3) Întrerupere la sfârșit de bloc.

Trebuie menționat faptul că linia $\overline{\text{BUSREQ}}$ a circuitelor DMA este cu drenă în gol și bidirecțională: ca ieșire, lansează o cerere spre Z80 în vederea cedării magistralelor; ca intrare, sesizează momentul în care un alt circuit DMA cere acces la magistrale, pe care-l blochează până când circuitul DMA activ termină operația.

Linia $\overline{\text{INT}}/\overline{\text{PULSE}}$, cu drenă în gol, are și o a doua funcțiune, în afară de solicitarea de întreruperi și anume aceea de a genera spre un dispozitiv extern impulsuri modulo 256 octeți transferați. Aceste impulsuri nu sunt interpretate de Z80 drept cereri de întrerupere, deoarece sunt emise când circuitul DMA are controlul magistralelor și deci UCP este suspendată. Începutul unui interval de 256 de octeți poate fi decalat, prin program, la pornire, cu 1÷255 octeți.

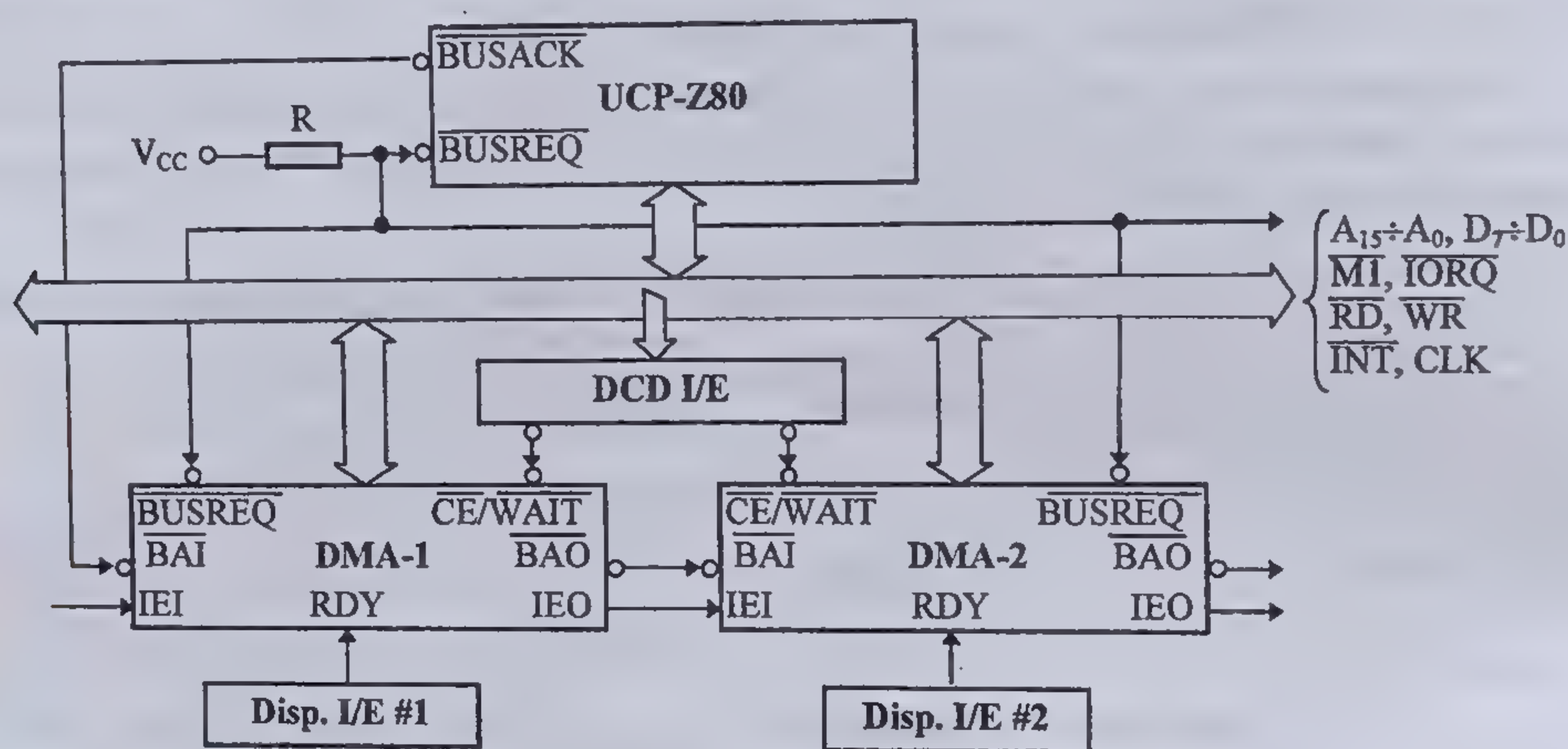


Fig.5.31. Conectarea mai multor dispozitive DMA-Z80

Linia $\overline{\text{CE}}/\overline{\text{WAIT}}$ are, de asemenea, funcționalitate dublă: în regim de slave este utilizată ca linie de selecție dispozitiv, iar în regim de master are rolul de sincronizare cu memorii sau dispozitive I/E mai lente. Mai mult, tot pentru sincronizarea vitezei de transfer cu dispozitive lente, pot fi programate întârzieri de 2, 3 sau 4 cicluri de tact, independent pentru dispozitivele sursă și destinație (*ciclu variabil*), caz în care logica externă de așteptare poate lipsi.

Adresele DMA conținute de cele două porturi, A și B (fig.5.30), sunt multiplexate pe magistrala de adrese a sistemului, după cum circuitul DMA citește date din dispozitivul sursă, respectiv le înscrie în dispozitivul destinație. Adresa curentă a fiecărui dispozitiv este păstrată în două registre de adresă de 16 biți, care pot fi citite de UCP. Adresele generate pot fi fixe sau incrementate/decrementate automat, începând de la adresa de start, înscrisă la programare. Posibilitatea de a genera o adresă fixă elimină necesitatea semnalelor de selecție a porturilor I/E. De asemenea, circuitul DMA-Z80 are facilitatea *autorestart*, prin care adresa de start a fiecărui dispozitiv poate fi reîncărcată automat, la terminarea unui transfer. Numărătorul de octeți este șters în momentul reîncărcării adresei.

Pentru citirea datelor este utilizată o schemă rapidă de tip "conductă" (pipeline), prin care operațiile asupra unui octet nu se termină până când nu este citit și octetul următor. Rezultă că numărul minim de octeți într-un bloc este 2, iar dimensiunea blocului înscrisă în numărătorul de octeți trebuie să fie N-1, N fiind lungimea blocului (maximum 64 Koct).

5.4.2. Programarea circuitului DMA-Z80

În vederea gestionării multiplelor sale facilități, circuitul DMA-Z80 posedă un set de 28 de registre, din care 21 sunt registre de comandă, ce pot fi numai înscrise și 7 registre de stare, care pot fi numai citite de UCP. Toate registrele au capacitatea de 8 biți, iar informația de 16 biți se depune în registre adiacente.

Registrele de comandă sunt organizate în 7 grupuri, denumite registre de înscriere și notate WR0 - WR6 (Write Register). Fiecare grup, cu excepția registrului WR5, are un registru de bază și unul sau mai multe registre secundare asociate, după cum urmează:

- WR0 - registru de bază;
 - LSB al adresei de start pentru portul A;
 - MSB al adresei de start pentru portul A;
 - LSB al lungimii blocului;
 - MSB al lungimii blocului;
- WR1 - registru de bază;
 - registru secundar pentru ciclu variabil la portul A;
- WR2 - registru de bază;
 - registru secundar pentru ciclu variabil la portul B;
- WR3 - registru de bază;
 - registru secundar pentru mască;
 - registru secundar pentru octetul de coincidență;
- WR4 - registru de bază;
 - LSB al adresei de start pentru portul B;
 - MSB al adresei de start pentru portul B;
 - registru pentru controlul întreruperilor;
 - registru pentru controlul impulsurilor;
- WR5 - registru de bază;
- WR6 - registru de bază;
 - registru secundar pentru masca de citire a registrelor de stare.

Registrele de bază din fiecare grup conțin biți de identificare (D_0 , D_1 și D_7), biți de comandă și biți indicatori (pointeri), pentru a selecta registrele secundare ale grupului. Pentru a scrie într-un registru secundar este necesar să fie înscris mai întâi un octet în registrul de bază, care să selecteze registrul respectiv.

Pentru exemplificare, în fig.5.32 este prezentat grupul WR0. Biții D_6+D_3 selectează, pe "1", registrele secundare care urmează să fie încărcate, biții D_1 și D_0 stabilesc modul de lucru, iar prin bitul D_2 se poate stabili sensul de transfer DMA. Detalii despre

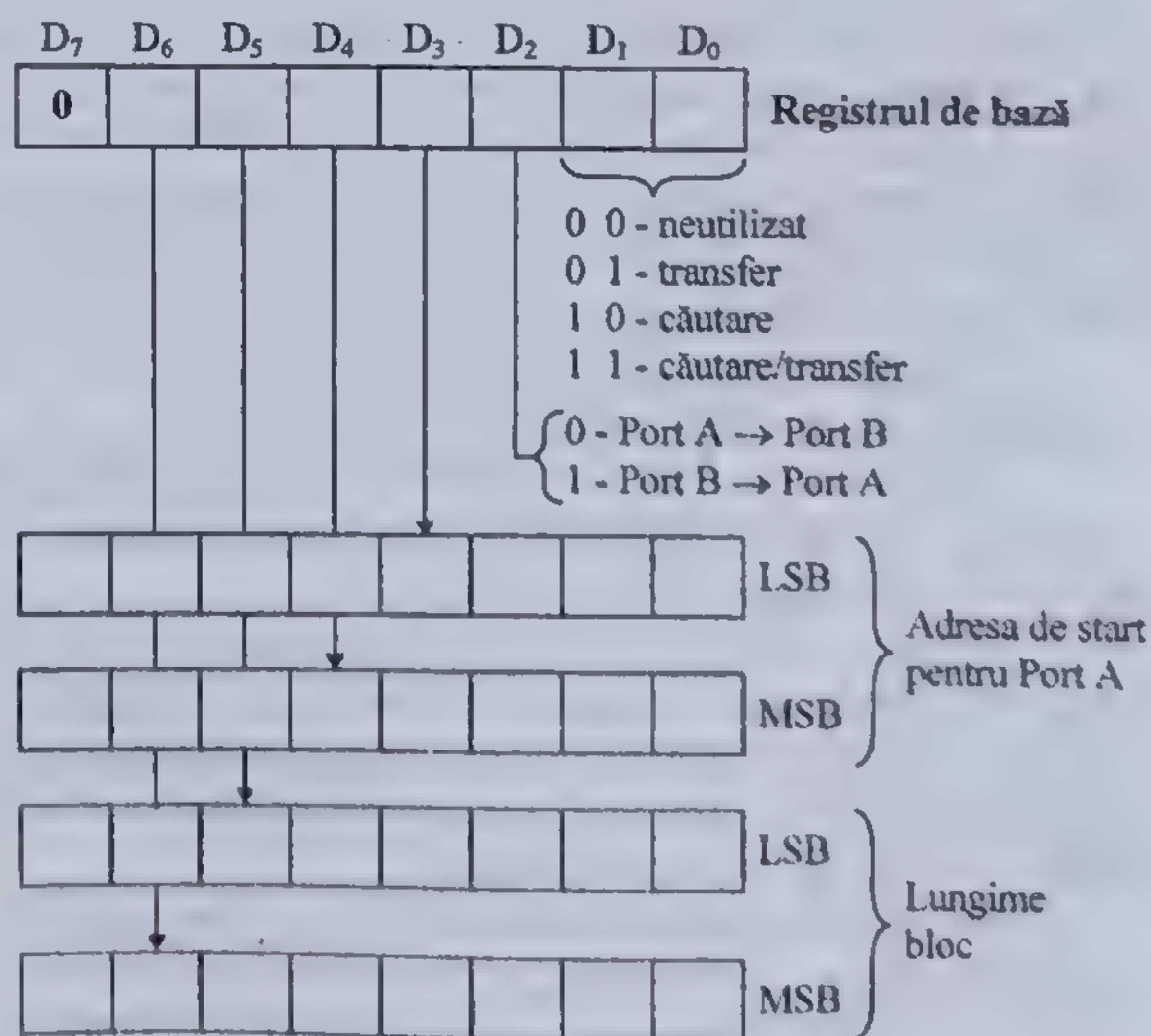


Fig.5.32. Structura grupului de registre de comandă WR0

celelalte grupuri de registre de comandă se găsesc în [10, 44, 45].

Registrele de stare sunt notate RR0÷RR6 (Read Register), nu au registre secundare asociate și conțin următoarele informații:

- RR0 - octet de stare;
- RR1 - LSB al număratorului de octeți;
- RR2 - MSB al număratorului de octeți;
- RR3 - LSB al adresei din portul A;
- RR4 - MSB al adresei din portul A;
- RR5 - LSB al adresei din portul B;
- RR6 - MSB al adresei din portul B.

În fig.5.33 este prezentat conținutul registrului de stare RR0. Pentru celelalte registre conținutul este evident, din cele arătate mai sus.

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	
x	x				x			RR0
								1 - A apărut funcționarea DMA
								0 - RDY este activ
								0 - Există o întrerupere în așteptare
								0 - S-a detectat coincidența
								0 - S-a detectat sfârșitul blocului

Fig.5.33. Configurația octetului de stare citit din RR0

Programarea circuitului DMA-Z80 se realizează de către UCP, înaintea oricărui transfer sau căutări de date, adresându-l ca port I/E și transmițându-i o secvență de octeți de comandă prin intermediul instrucțiunilor de ieșire (de ex. OTIR).

După conectarea tensiunii de alimentare, la inițializare sau în momentul în care UCP scrie în registrele de comandă, circuitul DMA-Z80 trece într-o stare de inactivitate. În această stare, circuitul nu poate solicita cedarea magistralei sistemului.

După programare, printr-o comandă de activare (înscrișă în registrul de bază al WR6), circuitul poate obține controlul magistrelor sistemului și poate efectua transferul/căutarea de date DMA. Practic, prin registrul de bază al grupului WR6 pot fi comandate diferite regimuri ale controlerului, cum ar fi: inițializarea (C3h), încărcarea (CFh), activarea generării întreruperilor (ABh), dezactivarea acestora (AFh), validarea după o instrucțiune RETI (B7h), citirea octetului de stare (BFh), inițierea secvenței de citire a registrelor de stare (A7h), activarea DMA (87h), dezactivarea DMA (83h) ș.a.

Microprocesorul UCP poate transmite comenzi unui circuit DMA în ambele stări, activă sau inactivă, acesta trecând automat în stare inactivă.

Înscrierea octeților de comandă la programare se face într-unul sau mai multe grupuri de registre de înscriere (WR0÷WR6), în corespondență cu modul de funcționare și tipul de transfer/căutare dorite.

Întotdeauna se începe cu octetul pentru registrul de bază al unui grup și se continuă cu registrele secundare asociate, specificate în octetul de bază. De exemplu, dacă în registrul de bază al grupului WR0 (v.fig.5.32), pe lângă biții de identificare D₇, D₁ și D₀, biții D₅, D₄ și D₃ sunt poziționați în "1" iar D₆ în "0", următorii trei octeți vor fi depuși în portul A (adresa de start) și în registrul inferior al lungimii blocului.

Citirea registrelor RR0÷RR6 se face de către UCP, prin adresarea controlerului ca port I/E, cu instrucțiuni de intrare (de ex. INIR). Citirea se realizează întotdeauna în secvență fixă, începând cu RR0 și încheind cu RR6. Dacă prin registrul secundar al grupului WR6 sunt mascate

unele registre RR, atunci secvența de citire va obține numai conținutul registrelor care nu sunt excluse de octetul de mască.

Exemplu: Să se scrie secvențele de inițializare și citire stare pentru un circuit DMA desemnat prin adresa simbolică DMAC. Cuvintele de comandă se află memorate într-un tabel, la adresa cmddma, de lungime lcmd. Conținutul registrelor RR0÷RR6 se va depune în memorie începând de la adresa stadma, tabelul având dimensiunea lmare.

; RUTINA DE ÎNIIALIZARE CONTROLER DMA-Z80

```
inidma: LD      HL,cmddma      ; Adresă tabel comenzi DMA.
        LD      C,DMAC        ; Adresă circuit DMA.
        LD      B,lcmd        ; Numărul octeților de comandă.
        OTIR                     ; Încarcă cuvintele de comandă în controlerul DMA.
        RET
```

; RUTINA DE CITIRE REGISTRE DE STARE DMA

```
readma: LD      HL,stadma      ; Adresa tabelului în care vor fi depuși octeții de stare.
        LD      C,DMAC        ; Adresa circuitului DMA-Z80.
        LD      B,lmare       ; Numărul de registre de stare care vor fi citite.
        LD      A,0A7h        ; WR6 - comandă inițializare secvența de citire stare
        OUT     (DMAC), A
        INIR                     ; Citește registrele de stare.
        RET
```

Ambele subrutine pot fi apelate din programul principal, după înscrierea parametrilor în cmddma, pentru a efectua transferul DMA dorit. La rutina inidma nu apare programarea vectorului de întrerupere deoarece acesta se transmite din tabel, prin WR4. În rutina readma nu s-a mascat nici un registru de stare, deci vor fi citite toate cele 7 registre. Dacă nu se dorește citirea unor registre, trebuie transmis un cuvânt de mascare înaintea începerii secvenței de citire.

5.5. Interfațarea cu periferice în sistemele cu Z80

La interfațarea cu periferice se utilizează mult porturile I/E paralele ale familiei Intel, în special circuitul PPI 8255. Dar, în acest caz, nu sunt folosite complet facilitățile microprocesorului Z80, ceea ce se poate realiza numai prin utilizarea interfeței paralele PIO-Z80.

5.5.1. Circuitul de interfațare paralelă PIO-Z80

Circuitul PIO-Z80 (*Parallel Input/Output Controller*), direct compatibil cu UCP-Z80, permite interfațarea paralelă cu perifericele a sistemelor bazate pe acest microprocesor. Transferul de date între UCP și periferice poate utiliza întreruperile în modul 2. O caracteristică aparte a circuitului PIO este posibilitatea de a întrerupe unitatea centrală la apariția unor condiții de stare specificate, la periferic, cum ar fi anumite condiții de alarmă. În acest mod se reduce semnificativ timpul necesar UCP pentru interogarea perifericelor.

Circuitul PIO-Z80 are două porturi de 8 biți (Port A și Port B), linii dedicate pentru controlul transferului, precum și toate elementele necesare conectării directe la magistralele microprocesorului Z80. În fig.5.34 se prezintă schema bloc internă a PIO-Z80, împreună cu liniile specifice de conectare la sistem și la periferice.

Interfața cu UCP dispune de un tampon bidirecțional de 8 biți pentru conectarea PIO la magistrala de date (D_7+D_0) și de linii de comandă specifice UCP-Z80: $\overline{M1}$, \overline{IORQ} , \overline{RD} (intern este generat și \overline{WR}). De asemenea, dispune de linii de control a tipului de informație ce urmează a fi vehiculată pe magistrala D_7+D_0 , C/\overline{D} (Control or Data select), și pentru selecția porturilor: B/\overline{A} (Port A or B select). Obișnuit, cele două linii se conectează la liniile A_1 și respectiv A_0 ale

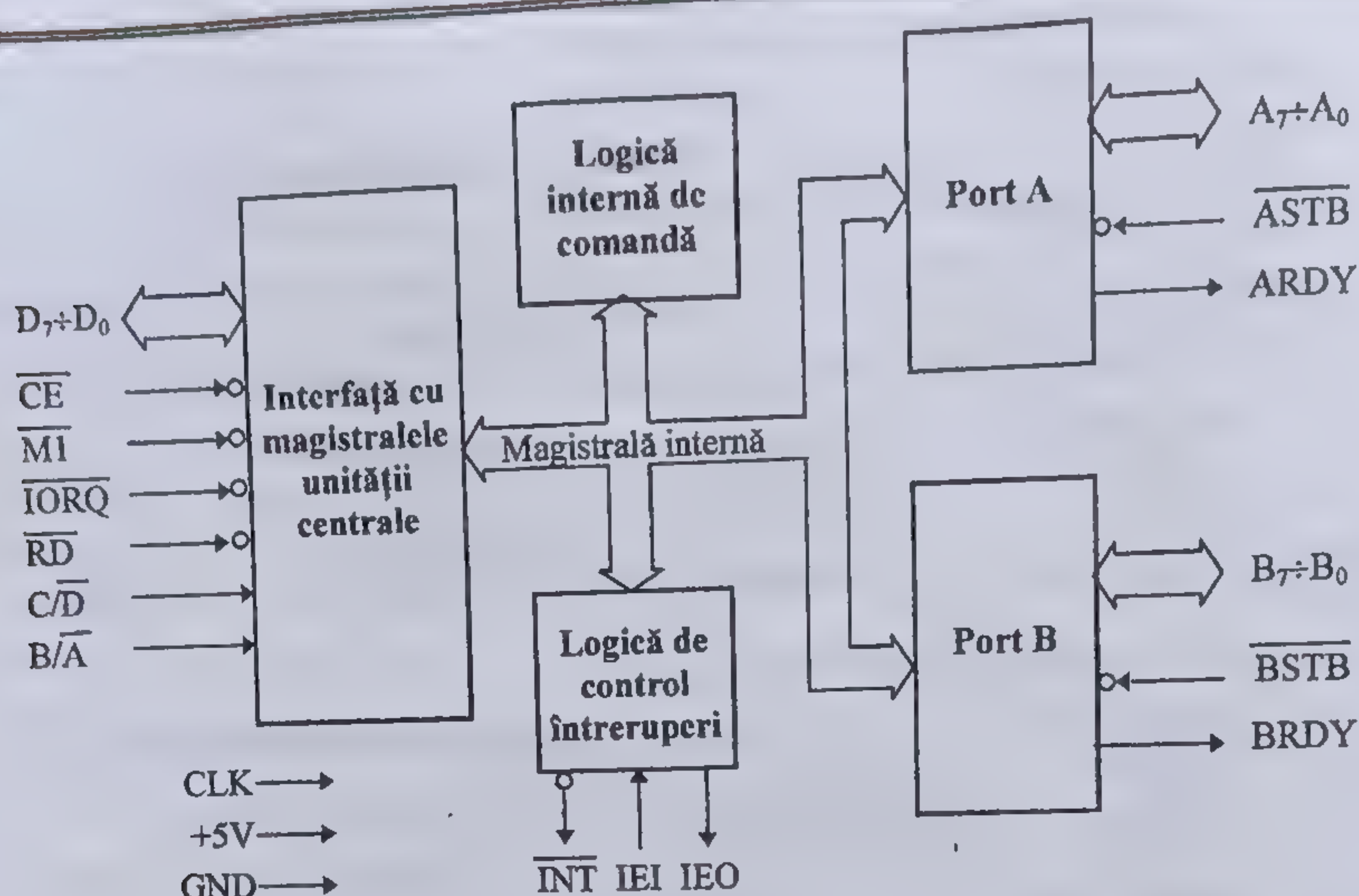


Fig.5.34. Schema bloc internă a circuitului PIO-Z80

magistralei de adrese. Împreună cu linia de activare a dispozitivului, \overline{CE} , semnalele de comandă menționate au efectul arătat în tab.5.10.

Tab.5.10.

\overline{CE}	$\overline{M1}$	\overline{IORQ}	\overline{RD}	C/\overline{D}	B/\overline{A}	Efect
0	1	0	0	0	0	Citire date din portul A
0	1	0	1	0	0	Înscriere în portul A
0	1	0	0	0	1	Citire date din portul B
0	1	0	1	0	1	Înscriere date în portul B
0	1	0	1	1	0	Înscriere comenzi în portul A
0	1	0	1	1	1	Înscriere comenzi în portul B
x	0	0	1	x	x	Identificare solicitant întrerupere

Cele două porturi, A și B, au registre distincte de intrare și de ieșire, precum și registre pentru programarea transferului I/E (fig.5.35). Ieșirile portului B pot comanda tranzistoare Darlingtion, generând 1,5mA (max. 3,8mA) la 1,5V.

Perifericul se conectează la liniile $P_7 \div P_0$, iar transferul cu confirmare se desfășoară sub controlul semnalelor READY și \overline{STROBE} . Modul de lucru al unui port este memorat într-un registru de 8 biți (MW), din care numai cei mai semnificativi doi biți sunt folosiți.

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	
M1	M0	x	x	1	1	1	1	MW
				Biți de identificare MW				
0	0	- Modul 0						
0	1	- Modul 1						
1	0	- Modul 2						
1	1	- Modul 3						

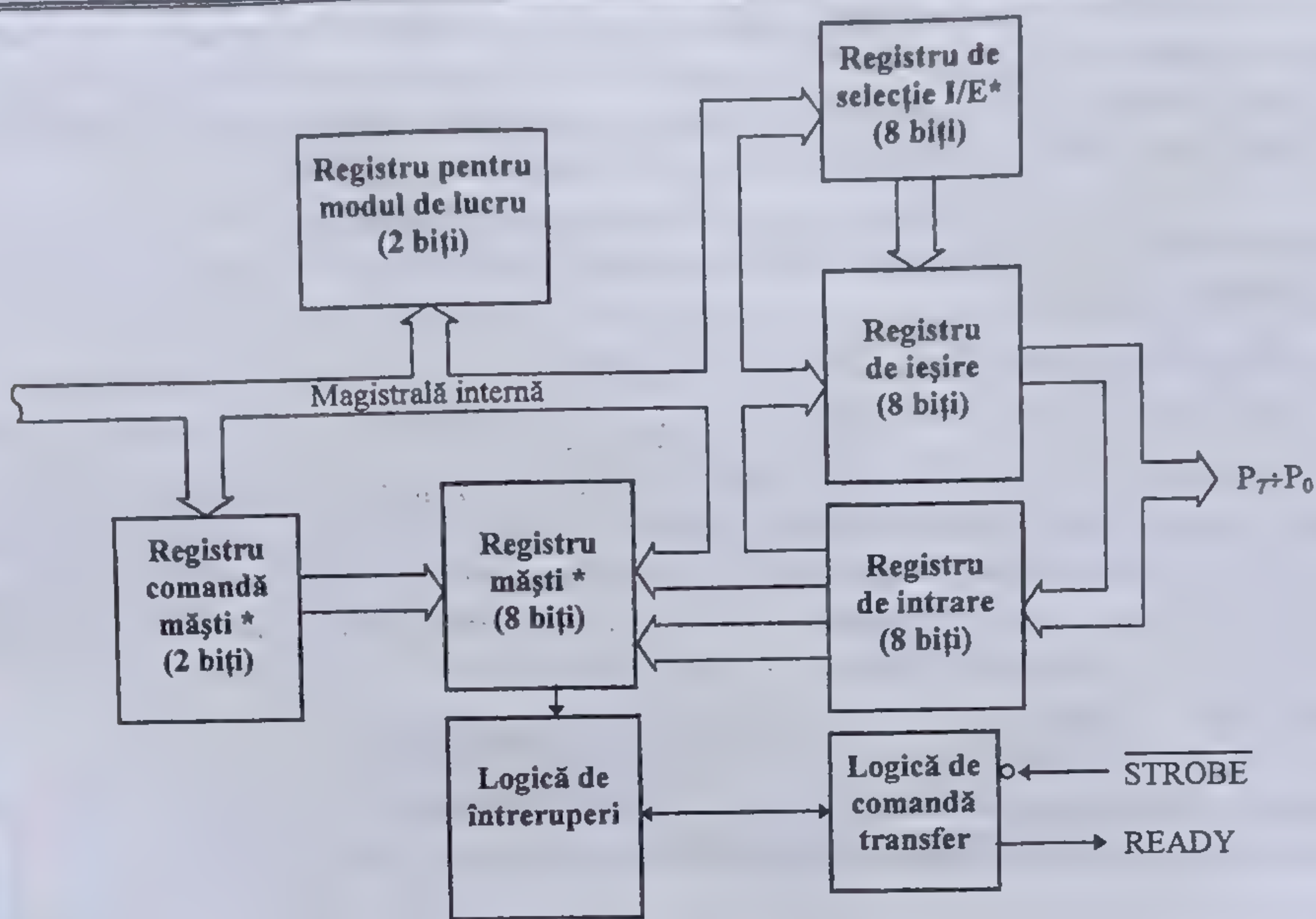


Fig.5.35. Structura internă a unui port I/E (A sau B); *) registre utilizate numai în modul 3

Dacă cel puțin unul din porturi lucrează cu întreruperile, la programarea circuitului este necesar să se înscrie un vector de întreruperi (VI).

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	
V ₇	V ₆	V ₅	V ₄	V ₃	V ₂	V ₁	0	VI
Biți stabiliți la programare							Identifică VI	

Mai mult, în modul 3 poate fi programată și logica după care portul emite întreruperea (condiție AND/OR între liniile active), nivelul logic activ, precum și masca dorită. Pentru specificarea celor arătate mai sus este necesar să se transmită logicii de întreruperi un cuvânt de comandă întreruperi (ICW1).

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	
EI	AND / OR	LEVEL	MASK	0	1	1	1	ICW1
				Identifică ICW1				
				1 - Urmează octetul de mască				
				1/0 - Nivelul activ al liniilor nemascate				
				1 - AND; 0 - OR				
				1 - Validare generare întreruperi; 0 - invalidare				

Dezactivarea temporară a generării întreruperilor în timpul funcționării, fără modificarea celorlalți parametri transmiși anterior prin ICW1, se poate realiza prin înscrierea unui alt cuvânt de control întreruperi, ICW2.

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	
$\overline{\text{DI}}$	x	x	x	0	0	1	1	ICW2
				Identifică ICW2				
				0/1 - Dezactivare/activare întreruperi				

La transmiterea acestor comenzi trebuie să se țină cont de asincronismul evenimentelor produse pe liniile $P_7 \div P_0$ față de funcționarea UCP. Pentru evitarea erorilor este necesară dezactivarea temporară a întreruperilor microprocesorului, pe durata transmiterii comenzilor de dezactivare a întreruperilor la PIO-Z80, similar celor arătate și la CTC-Z80 (v. §5.3.1.2).

În cazul în care bitul $D_4=1$ în cuvântul de comandă întreruperi ICW1, atunci trebuie transmis un octet registrului de măști (MSKW). Biții "1" înscrisi în acest registru maschează liniile corespunzătoare ale portului.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0	MSKW

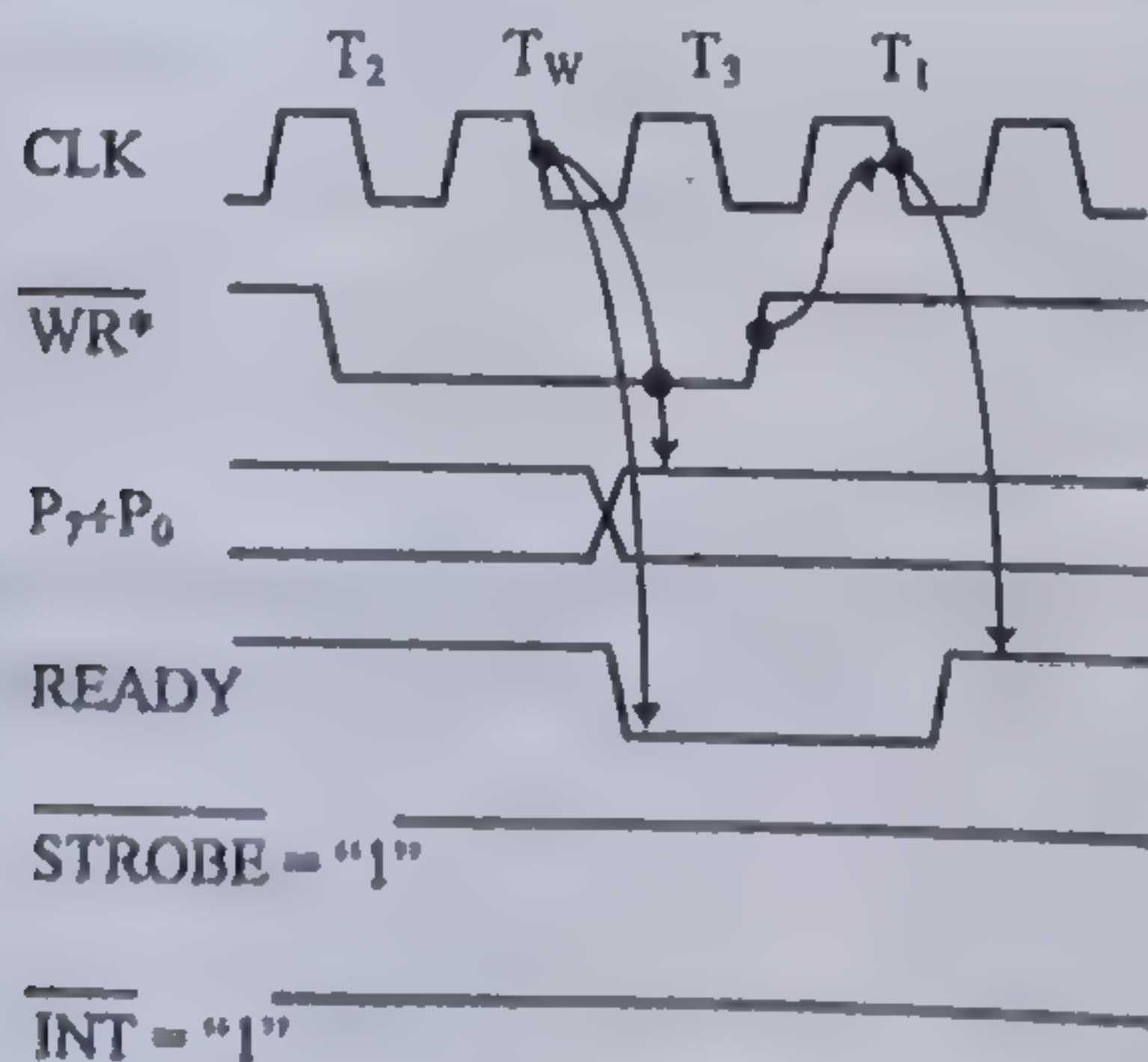
În condiția de generare a unei întreruperi vor conta numai liniile nemascate care sunt orientate ca linii de intrare. Specific numai modului 3 de funcționare, această orientare individuală a liniilor I/E ale unui port se face prin intermediul registrului de selecție I/E (IOW). După cum un bit din acest registru se poziționează în "1" sau "0", linia corespunzătoare a portului se configurează ca linie de intrare, respectiv de ieșire.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
I/\bar{E}_7	I/\bar{E}_6	I/\bar{E}_5	I/\bar{E}_4	I/\bar{E}_3	I/\bar{E}_2	I/\bar{E}_1	I/\bar{E}_0	IOW
1 - linie de intrare; 0 - linie de ieșire								

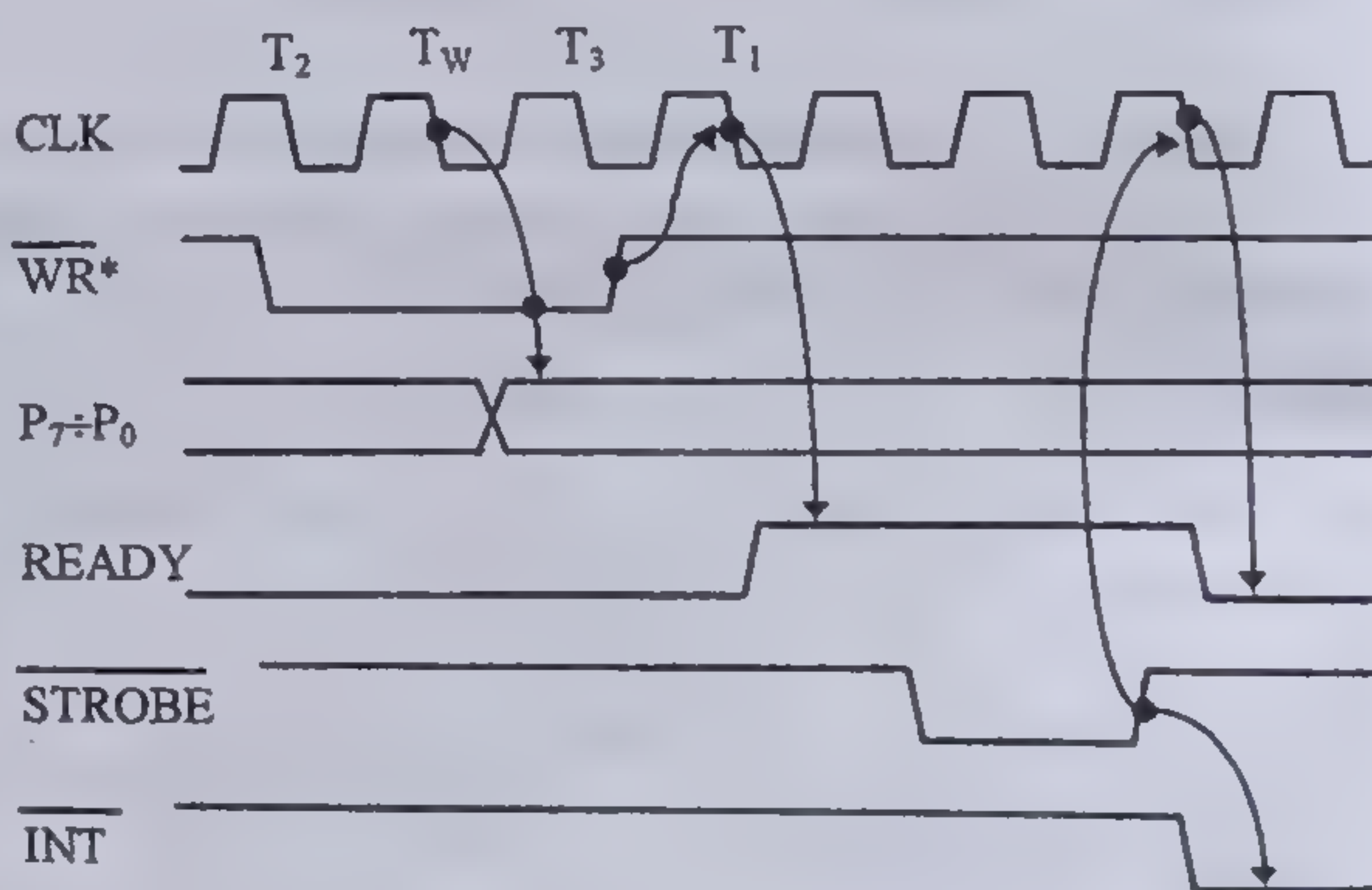
5.5.2. Modurile de lucru ale circuitului PIO-Z80

Acest circuit poate fi programat să funcționeze în patru moduri (vezi MW), și anume: *ieșire pe octet* (modul 0), *intrare pe octet* (modul 1), *bidirecțional pe octet* (modul 2) și *intrare/ieșire pe bit* (modul 3).

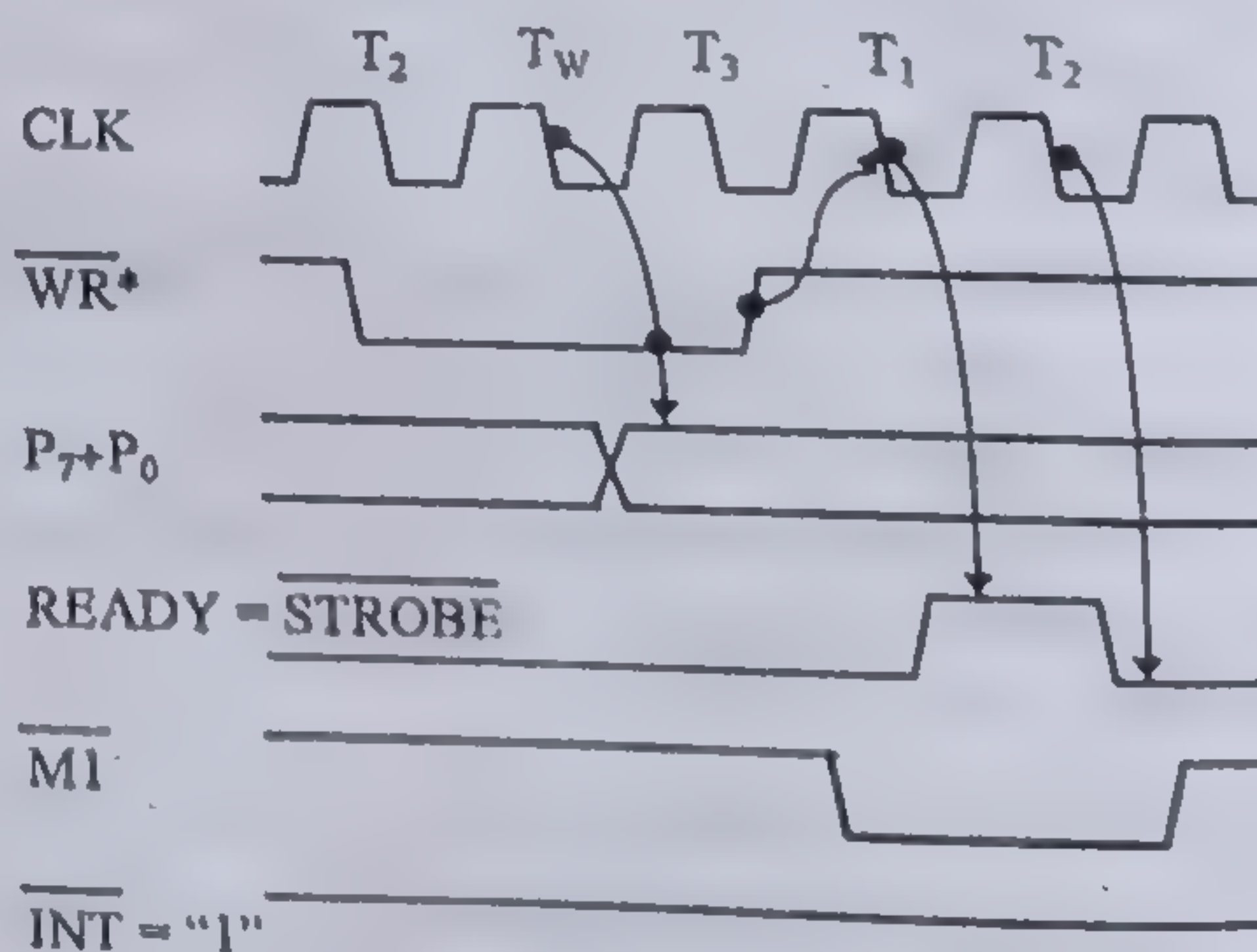
În modul 0, oricare din porturile A și B poate fi programat ca port de ieșire. Când un octet de date este înscris într-un port, acesta apare pe



b) Linia \overline{STROBE} este fixată la "1".



a) Se utilizează ambele linii, READY și \overline{STROBE} .



c) Linia READY conectată la linia \overline{STROBE} .

Fig.5.36. Cronogramele în modul 0 de funcționare al circuitului PIO-Z80 (a, b, și c)

Partea a II-a - Sisteme cu microprocesoare de 8 biți

liniile $P_7 \div P_0$, iar ieșirea READY se activează și indică perifericului asociat că octetul este disponibil. La rândul lui, perifericul confirmă preluarea octetului prin activarea semnalului $\overline{\text{STROBE}}$. În acest moment, dacă întreruperile sunt activate, se generează o cerere de întrerupere spre UCP. Funcționarea circuitului este sincronizată cu semnalul de tact al sistemului (Φ) și respectă condițiile de dialog ale microprocesorului Z80 cu porturile I/E (fig.5.36a).

Registrele de ieșire ale porturilor pot fi încărcate în orice moment de către microprocesor, prin instrucțiuni de tip OUT. Semnalul $\overline{\text{WR}}^*$, generat intern în timpul unei astfel de instrucțiuni, validează înscrierea datelor de pe magistrala de date ($D_7 \div D_0$) în registrul de ieșire al portului selectat și determină apariția lor pe liniile $P_7 \div P_0$. După dezactivarea semnalului $\overline{\text{WR}}^*$, cu frontul descrescător al semnalului CLK se activează linia READY, care se menține în "1" până la dezactivarea semnalului $\overline{\text{STROBE}}$. Abia în acest moment se va activa linia $\overline{\text{INT}}$.

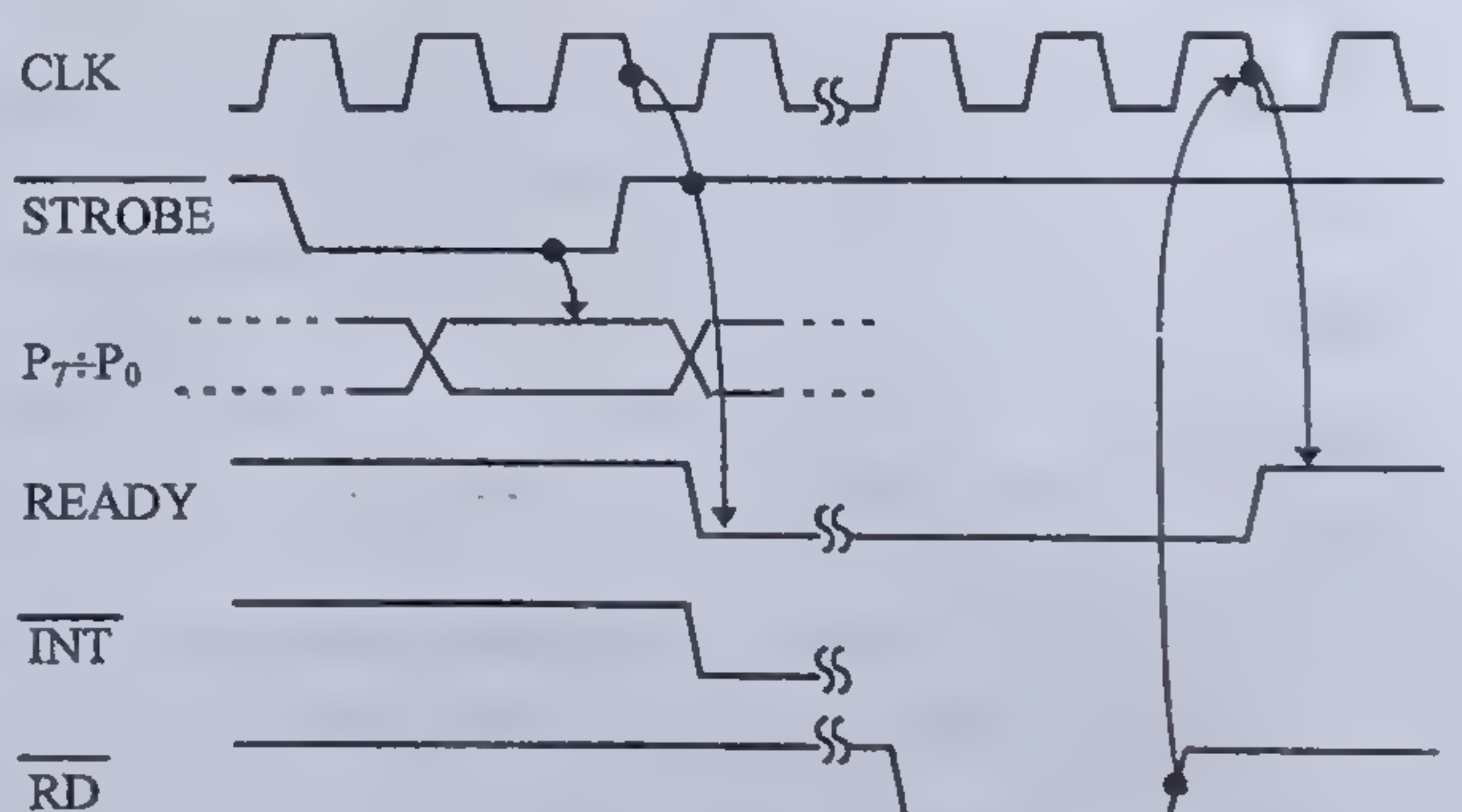
Transferul în modul 0 se poate realiza și fără utilizarea liniei $\overline{\text{STROBE}}$, când aceasta se conectează fie la "1" (fig.5.36b), fie împreună cu linia READY (fig.5.36.c). În ambele situații, se observă că nu se mai generează cereri de întrerupere.

Modul 1 este folosit pentru transferul informației de la periferice la UCP. Faptul că porturile sunt pregătite să primească informații este semnalizat perifericelor prin activarea semnalelor READY (ARDY, respectiv BRDY).

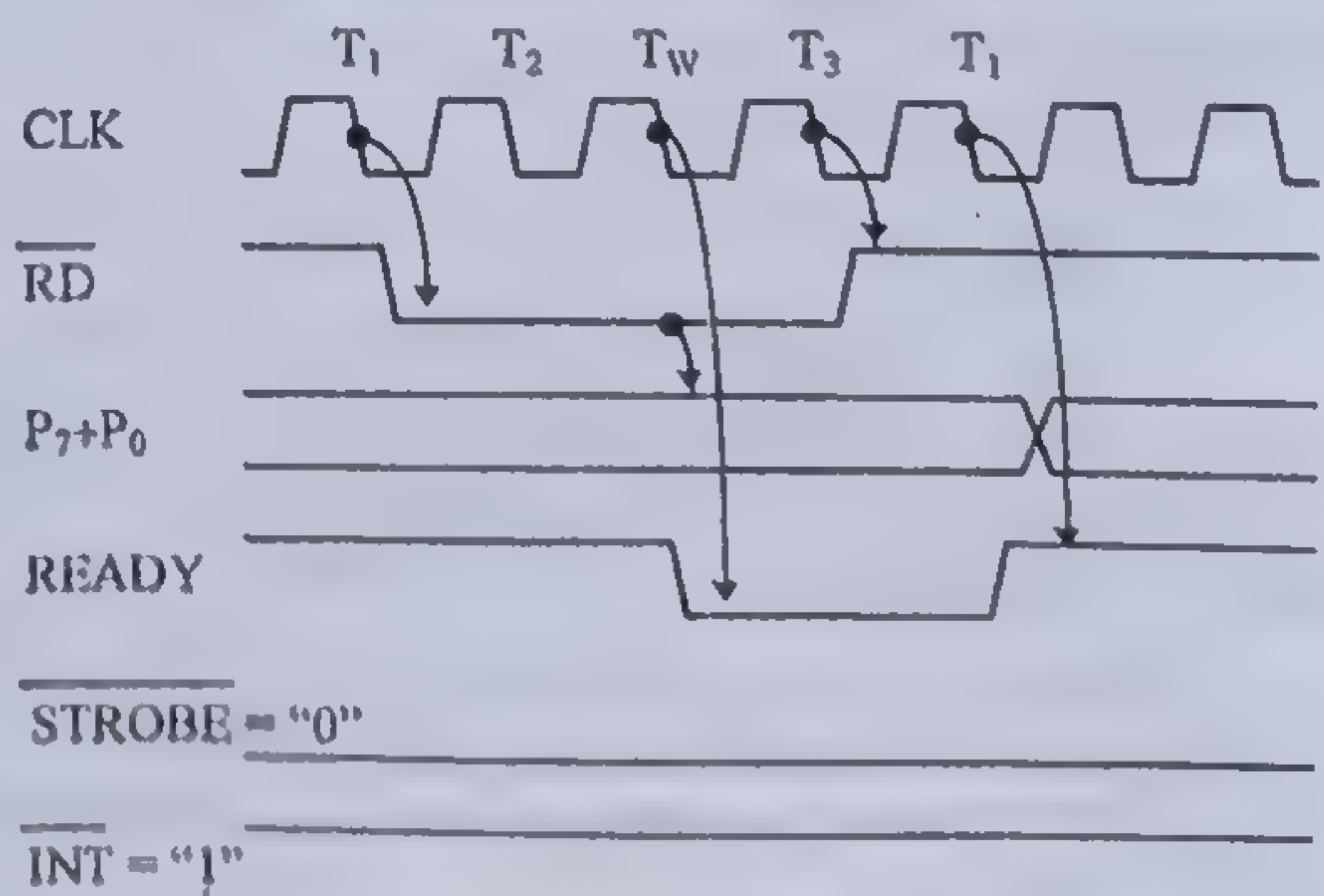
Un ciclu de intrare poate fi inițiat de către periferic prin activarea liniei $\overline{\text{STROBE}}$ asociată portului ($\overline{\text{ASTB}}$, respectiv $\overline{\text{BSTB}}$), așa cum se arată în fig.5.37a. În timp ce acest semnal este "0" logic, liniile de intrare ($P_7 \div P_0$) sunt eșantionate și octetul citit se transferă în registrul de intrare. Pe frontul pozitiv al semnalului $\overline{\text{STROBE}}$ se poate genera o cerere de întrerupere către UCP.

La revenirea semnalului $\overline{\text{STROBE}}$ în "1", odată cu primul front descrescător al tactului este dezactivat semnalul READY, care blochează transmiterea unui nou octet de către periferic. Octetul memorat în registrul de intrare este preluat de către UCP, printr-o instrucțiune de tip IN, de obicei în rutina de tratare a întreruperii. Semnalul READY va fi activat după citirea registrului de intrare al portului, pentru a preveni supraînscrierea unui nou octet.

Deoarece, la inițializarea circuitului PIO-Z80, ieșirile READY rămân pe "0", pentru a lansa un transfer în modul 1 este necesar să se execute o citire falsă,



a) Se utilizează ambele semnale, READY și STROBE.



b) Linia $\overline{\text{STROBE}}$ conectată la "0".

Fig.5.37. Diagramele de semnale la funcționarea circuitului PIO-Z80 în modul 1 (a și b)

care comandă trecerea liniilor READY în "1".

În cazul în care nu este utilizată, intrarea $\overline{\text{STROBE}}$ trebuie conectată la "0" și semnalele evoluează ca în fig.5.37b. Semnalul READY va fi dezactivat în timpul citirii registrului de intrare al portului, la $1\frac{1}{2}T$ după activarea liniei $\overline{\text{IORQ}}$, pentru a preveni supraînscriserea unui nou octet. Și în această situație, ca și la modul 0, nu se generează cerere de întrerupere.

În modul 2, bidirecțional, se utilizează numai portul A și semnalele de control al transferului de la ambele porturi. În acest caz, portul B trebuie programat obligatoriu în modul 3, cu toate liniile mascate.

O operație de ieșire este similară cu cea din modul 0, cu observația că datele din registrul de ieșire sunt depuse pe liniile $A_7 \div A_0$ numai pe durata activării semnalului $\overline{\text{ASTB}}$ (fig.5.38). Frontul crescător al acestui semnal poate fi utilizat de către periferic pentru a memora datele primite. O operație de intrare decurge asemănător cu cea din modul 1, dar folosește semnalele de comandă și întreruperea portului B.

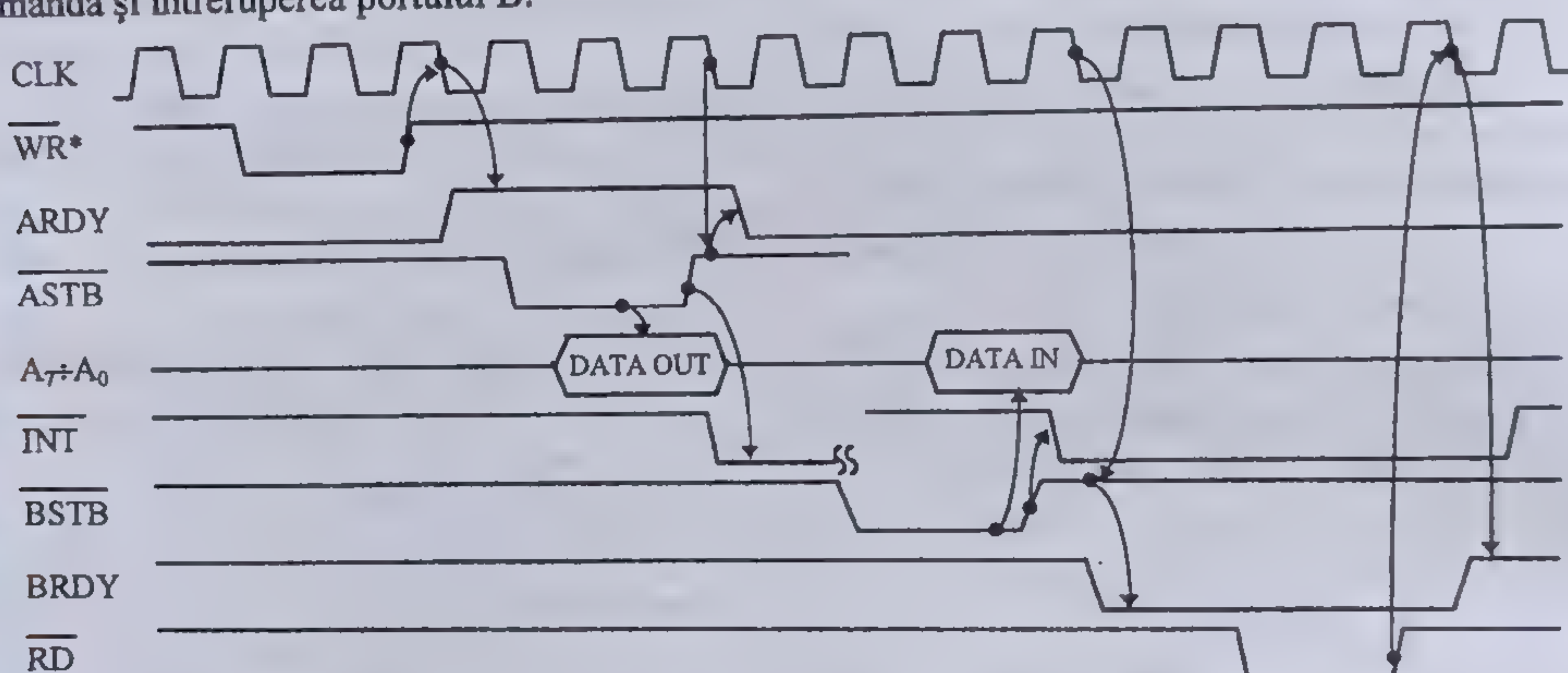


Fig.5.38. Diagrama de semnale la funcționarea circuitului PIO-Z80 în modul 2

În timpul unei operații de ieșire ($\overline{\text{ASTB}}$ activ), perifericul nu trebuie să inițieze o operație de intrare, deci semnalul $\overline{\text{BSTB}}$ trebuie să fie decalat față de $\overline{\text{ASTB}}$. Un alt conflict ar putea să apară atunci când, în rutina de întrerupere de intrare, se încearcă citirea octetului memorat în registrul de intrare în timp ce se desfășoară o operație de ieșire (linia $\overline{\text{ASTB}}$ activă). În acest caz, în locul octetului din registrul de intrare se obține octetul din registrul de ieșire. Evitarea unei astfel de situații se realizează prin efectuarea unei operații de ieșire ($\overline{\text{ASTB}} = 0$) numai dacă $\text{BRDY} = 1$, care indică faptul că octetul din registrul de intrare a fost citit.

În modul 3, liniile ambelor porturi pot fi definite individual prin program, fie ca ieșiri, fie ca intrări. Semnalele READY și $\overline{\text{STROBE}}$ aferente nu sunt utilizate. Liniile READY sunt aduse în "0" dacă ambele porturi se programează în modul 3. În cazul în care portul A este programat în modul 2 și portul B în modul 3, atunci BRDY are funcționalitatea descrisă mai sus.

Dacă se lucrează fără întreruperi, UCP poate să execute în orice moment operații de I/E. La citirea registrului de intrare, data citită este formată din biții din registrul de ieșire corespunzător liniilor de ieșire și din starea liniilor de intrare de dinaintea activării semnalului $\overline{\text{RD}}$ (fig.5.39).

Dacă se folosesc întreruperile, acestea se generează la îndeplinirea unei condiții logice AND/OR între liniile de intrare active ale portului, care nu sunt mascate. Atât condiția logică cât și

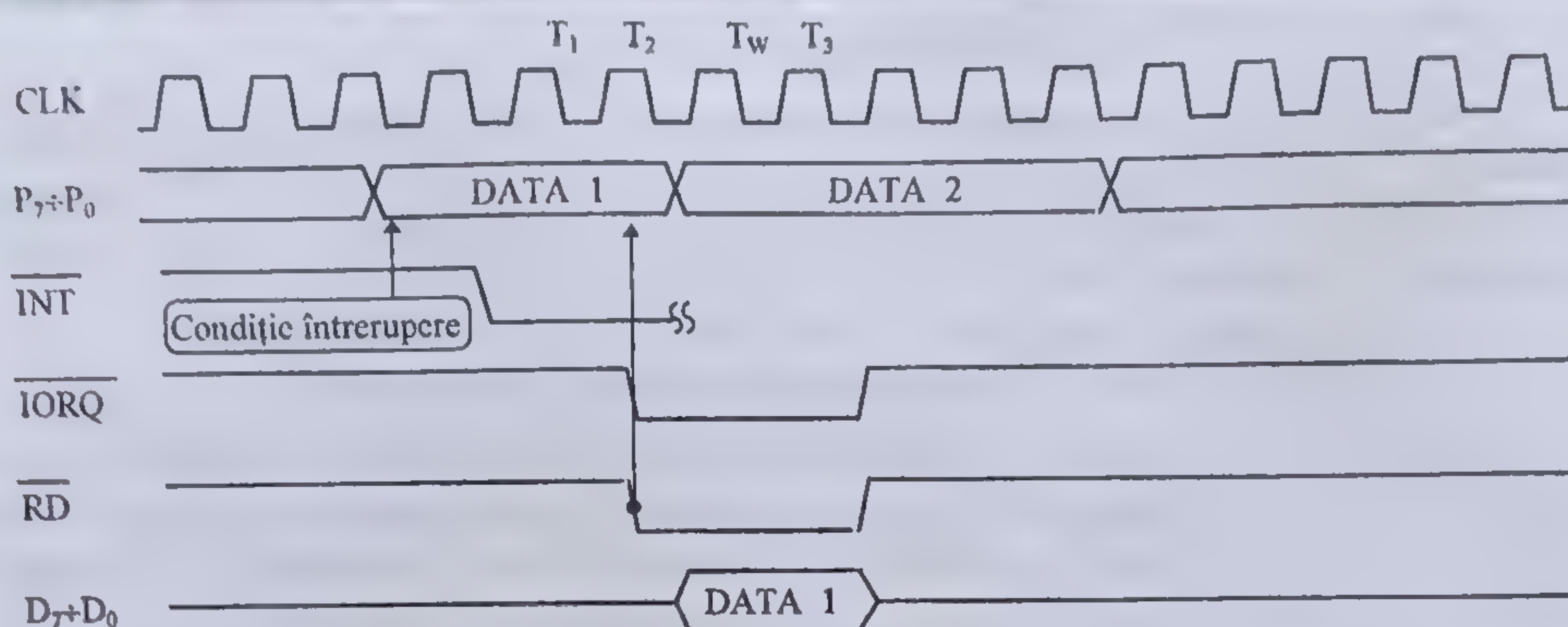


Fig.5.39. Citirea dintr-un port programat în modul 3

nivelul activ al liniilor de intrare se stabilesc la programare, prin cuvântul de comandă întreruperi ICW1.

Circuitul PIO-Z80 supraveghează permanent intrările nemascate ale portului, pentru a detecta îndeplinirea condiției programate. Atunci când această condiție se schimbă din "falsă" în "adevărată", se generează o cerere de întrerupere. Trebuie evidențiat faptul că, în cazul logicii OR, dacă două sau mai multe linii devin simultan active, se generează o singură cerere de întrerupere. Același lucru se produce dacă o linie devine activă în timp ce o altă linie era deja activă. Ambele situații sunt explicabile prin specificul logicii de tip "SAU-inclusiv".

La funcționarea în modul 3 cu întreruperi, proiectantul trebuie să țină cont și de următoarele restricții suplimentare:

- atât timp cât o întrerupere în curs de servire nu a fost (încă) achitată printr-o instrucțiune RETI, apariția unei noi tranziții a condiției de la "falsă" la "adevărată" nu poate genera o nouă cerere de întrerupere pentru același port.

- dacă condiția devine adevărată puțin înainte de sau după activarea semnalului $\overline{M1}$, cererea de întrerupere se generează după frontul pozitiv al acestui semnal, dacă condiția mai este adevărată. De aceea, condiția programată produce o cerere de întrerupere numai dacă rămâne adevărată un interval de timp mai mare decât durata activării semnalului $\overline{M1}$.

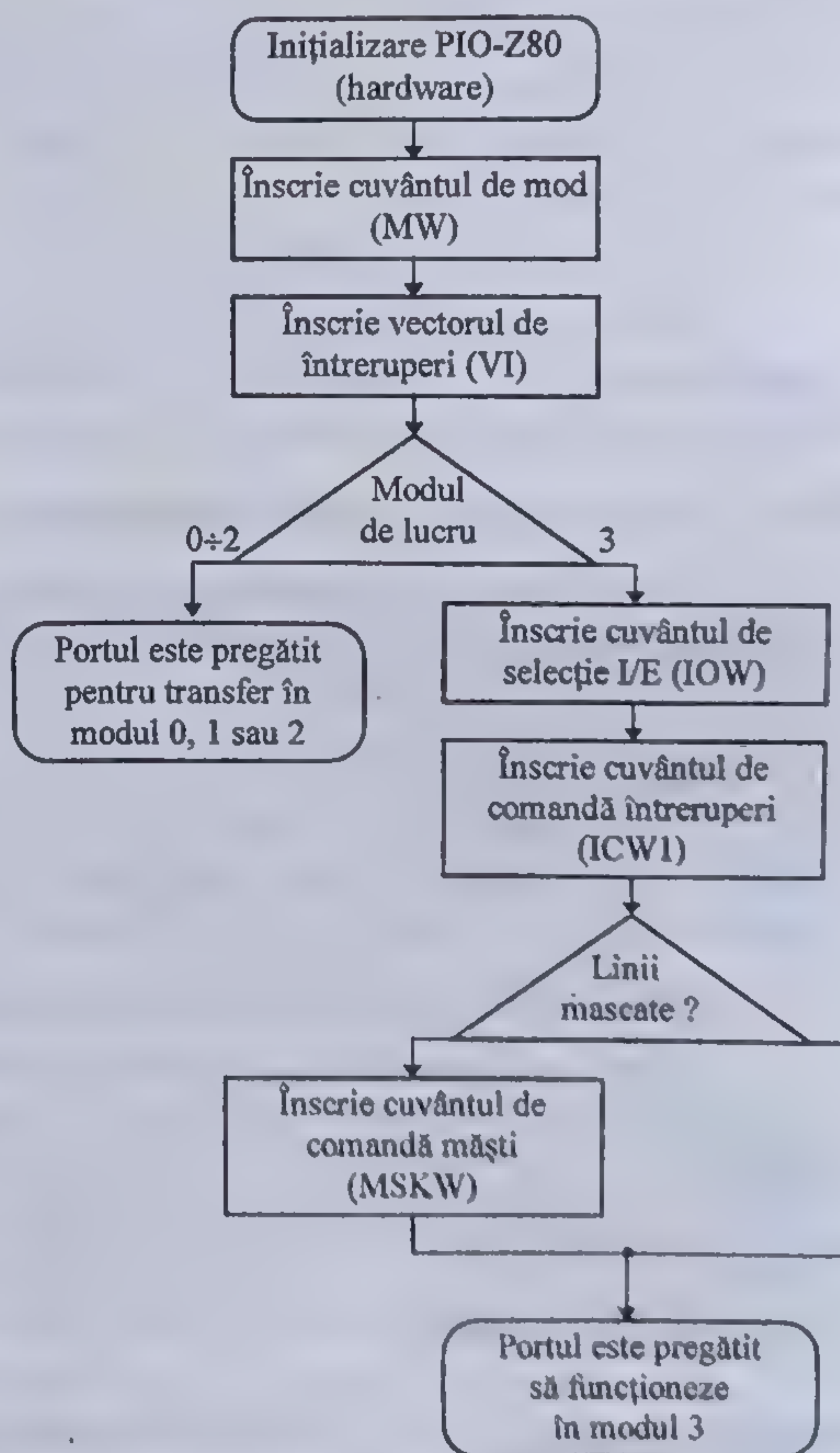


Fig.5.40. Secvența de programare a unui port al circuitului PIO-Z80

5.5.3. Programarea interfeței paralele PIO-Z80

La punerea sub tensiune sau la activarea liniei $\overline{M1}$, cu \overline{RD} și \overline{IORQ} pe "1", se produce inițializarea circuitelor PIO-Z80, cu următoarele efecte:

- selectarea modului 1 de lucru (intrare-octet), cu trecerea liniilor $A_7 \div A_0$ și $B_7 \div B_0$ în starea de înaltă impedanță și a ieșirilor ARDY și BRDY în "0" logic;
- forțarea registrelor de mască ale ambelor porturi în FFh;
- resetarea bistabililor de validare a întreruperilor celor două porturi.

Cea de a doua modalitate de inițializare (prin linia $\overline{M1}$), asigură suplinirea lipsei unei linii de RESET, impusă de limitarea la 40 de conexiuni externe. Obişnuit, pentru controlul liniei $\overline{M1}$ de la PIO-Z80 se foloseşte o poartă ŞI cu două intrări: una pentru comanda de inițializare, iar cealaltă pentru $\overline{M1}$ de la Z80.

Pentru funcționarea în modurile 0, 1 sau 2 este necesar ca, după inițializarea hardware, să se înscrie în fiecare port două cuvinte: cuvântul de comandă al modului de lucru (MW) și vectorul de întreruperi (VI). Este posibil ca transferul să fie realizat fără confirmare, similar modului 0 de la PPI 8255, caz în care se programează numai cuvântul de mod de lucru.

La funcționarea în modul 3, în afară de MW și VI, mai trebuie programat și registrul de selecție I/E (IOW) precum și un cuvânt de comandă al întreruperilor (ICW1), urmat eventual de un octet de mascare (MSKW).

Sintetic, pentru programarea unui port al unui circuit PIO-Z80 trebuie respectată secvența din fig.5.40. Structura cuvintelor este cea prezentată în §5.5.1.

După transmiterea cuvintelor corespunzătoare modului de lucru, UCP poate transmite sau recepționa date spre/de la porturi. În [10, 27, 28] sunt date exemple de interfațare a perifericelor cu PIO-Z80 și de programare a porturilor în cele 4 moduri de lucru.

5.6. Comunicația serială în sistemele cu Z80

Comunicația serială la sistemele bazate pe microprocesorul Z80 se realizează cu circuitul specializat SIO-Z80 (Serial Input/Output Controller). Circuitul face parte din familia Z80 și este destinat implementării aplicațiilor care necesită un transfer serial de date, pentru care dispune de două canale independente, A și B, full-duplex. Este o interfață programabilă, deosebit de versatilă, care înglobează atât funcțiile specializate de comunicație asincronă și sincronă (de tipul USART 8251, 8251A), precum și toate facilitățile necesare implementării protocoalelor de comunicație serială orientate pe bit (BOP) sau pe caracter (BCP). SIO-Z80 permite obținerea unor rate ridicate de comunicație de până la 500 Kbps, 800 Kbps la SIO-Z80A și 1,2 Mbps la SIO-Z80B.

5.6.1. Structura internă a circuitului SIO-Z80

Structura internă a circuitului SIO-Z80 este prezentată în fig.5.41. Este organizată în jurul unei magistrale interne, la care sunt conectate toate blocurile funcționale componente: interfața cu magistralele UCP, logica internă de comandă, logica de întreruperi, precum și blocurile componente ale celor două canale, A și B. Datorită limitării numărului de pini la 40, există trei variante de încapsulare: SIO-Z80/0 are semnalele \overline{TxCB} și \overline{RxCB} legate împreună (\overline{RxTxCB}), lui SIO-Z80/1 îi lipsește semnalul \overline{DTRB} , iar SIO-Z80/2 nu are \overline{SYNCB} .

Canalele de comunicație conțin registre de comandă și de stare, o logică de dialog cu modemul și blocuri pentru transmisia/recepția propriu-zisă. Schimbul de informații cu UCP se realizează cu ajutorul semnalelor cu care este dotată interfața cu magistralele sistemului, conform celor prezentate în tab.5.11.

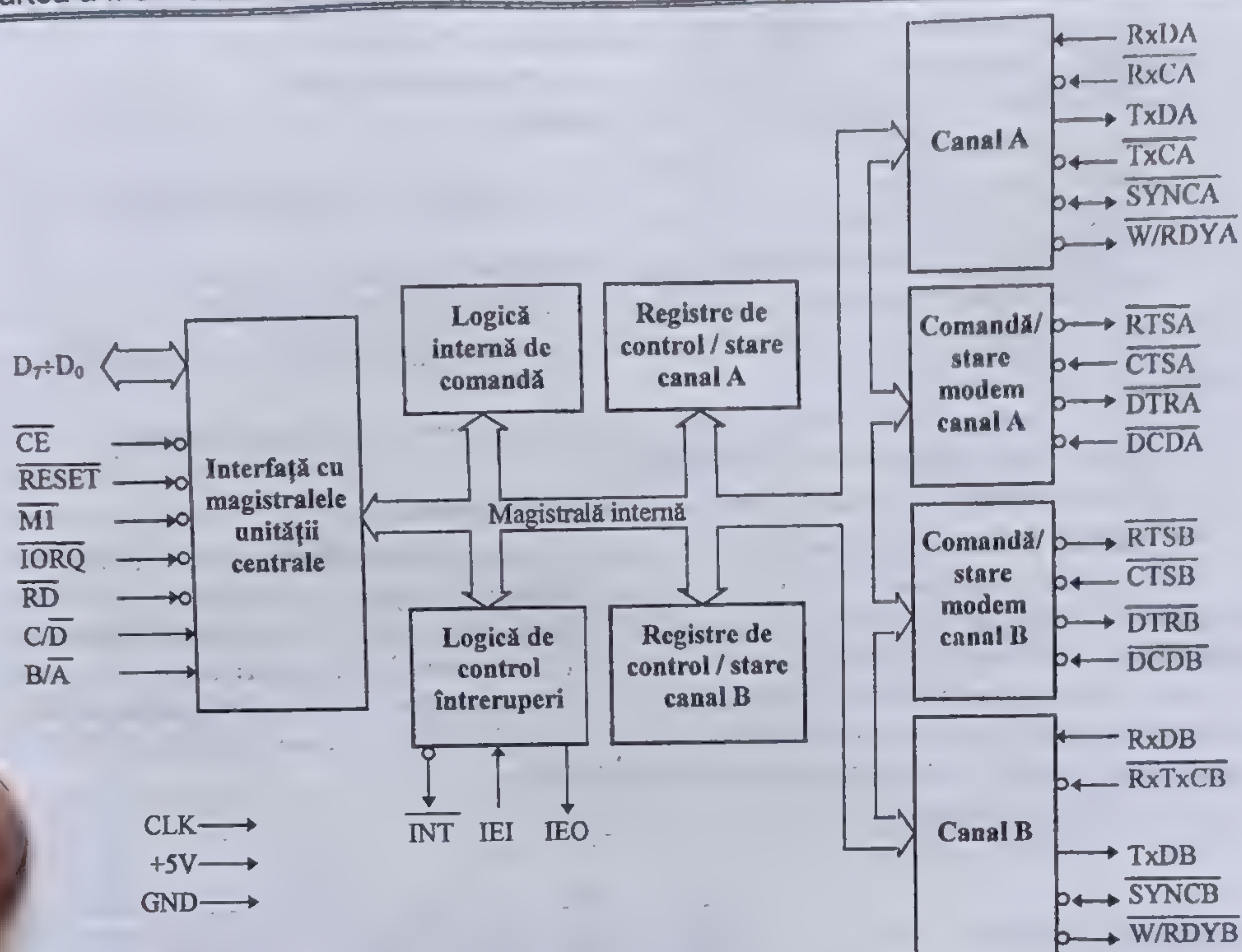


Fig.5.41. Schema bloc internă a circuitului SIO-Z80/0

Tab.5.11.

\overline{CE}	\overline{IORQ}	$\overline{M1}$	\overline{RD}	C/\overline{D}	B/\overline{A}	Acțiunea efectuată
0	0	1	0	0	0	Citire registru de date canal A
0	0	1	1	0	0	Scrisoare în registrul de date canal A
0	0	1	0	0	1	Citire registru de date canal B
0	0	1	1	0	1	Scrisoare în registrul de date canal B
0	0	1	0	1	0	Citire registru de stare canal A
0	0	1	1	1	0	Scrisoare în registrul de control canal A
0	0	1	0	1	1	Citire registru de stare canal B
0	0	1	1	1	1	Scrisoare în registrul de control canal B
x	0	0	1	x	x	Identificare solicitant întrerupere

Liniile C/\overline{D} (Control or Data select) și B/\overline{A} (Channel A or B select) asigură selectarea tipului datei vehiculate pe liniile D_7+D_0 , respectiv a canalului prin care are loc schimbul de informații. Aceste linii se conectează de obicei la liniile A_0 și respectiv A_1 ale magistralei de adrese a sistemului. Semnalul \overline{RESET} asigură inițializarea HW a circuitului, cu dezactivarea recepției/transmisiei în ambele canale (liniile TxD și semnalele de dialog cu modemul sunt forțate pe nivel "1" logic) și invalidează întreruperile.

După inițializarea HW, circuitul SIO-Z80 trebuie programat, prin înscrierea modului de funcționare și a parametrilor specifici în cele 8 registre de comandă ($WR0+WR7$), care pot fi doar înscrise:

TxDA, TxDB (Transmit Data) - ieșiri seriale de date, de nivel TTL;

TxCA, TxCB (Transmitter Clock) - intrări trigger-Schmitt, utilizate ca bază de timp pentru transmisie, similare celor de la recepție. Biții de date se depun pe liniile TxD, odată cu frontul căzător al semnalelor pe liniile TxC corespunzătoare.

SYNCA, SYNCB (Synchronization) - semnale a căror funcție depinde de modul de lucru. Astfel, în modul asincron sunt intrări și pot fi folosite, ca și liniile \overline{CTS} și \overline{DCD} , pentru dialog cu dispozitivele externe. Tranzițiile care au loc pe aceste intrări pot să întrerupă unitatea centrală dacă sunt validate întreruperile de stare/externe. Dacă nu sunt utilizate, nu trebuie lăsate în gol pentru a nu produce întreruperi de stare/externe parazite (în caz că acestea sunt validate).

În modul cu sincronizare externă, sunt intrări de sincronizare. Logica de sincronizare externă, după detecția caracterelor SYNC, trebuie să aștepte două perioade complete ale ceasului de recepție și apoi să activeze intrările \overline{SYNC} , pentru a începe recepția sincronă. Intrările trebuie să rămână activate până când unitatea centrală informează logica de sincronizare externă că s-a pierdut sincronizarea sau că poate să înceapă un nou mesaj.

În modul cu sincronizare internă, MONOSYNC (un singur caracter de sincronizare) sau BISYNC (2 caractere), pe liniile de ieșire \overline{SYNC} apare un impuls "0" ori de câte ori se recunoaște pe linia RxD o configurație de caractere de sincronizare.

W/RDYA, W/RDYB (Wait/Ready) - sunt ieșiri cu drenă în gol atunci când sunt programate pentru funcția WAIT, pentru a sincroniza unitatea centrală cu viteza de comunicație a SIO. Liniile sunt ieșiri atunci când sunt programate pentru funcția READY, necesară pentru a sincroniza circuitele DMA (8257, DMA-Z80) cu viteza de comunicație a SIO. La inițializare ieșirile sunt de tip drenă în gol, specifice funcției WAIT.

Blocurile stări și comenzi canal conțin semnalele necesare interfațării circuitului cu modemuri sau alte echipamente externe:

RTSA, RTSB (Request To Send) - cerere de transmisie. Acestea sunt linii de ieșire de uz general, activate atunci când bitul RTS din registrul de comandă WR5 este programat pe "1". Dacă se utilizează modul asincron și bitul respectiv este programat pe "0", liniile \overline{RTS} vor fi dezactivate la golirea tamponului de transmisie. În modul sincron, liniile \overline{RTS} urmăresc starea biților cu același nume din WR5.

CTSA, CTSB (Clear To Send) - pregătit pentru emisie. Dacă bitul D5 (autovalidare) din registrul WR3 se programează pe "1", activarea acestor intrări trigger-Schmitt validează circuitele de transmisie. Dacă bitul respectiv este programat pe "0", liniile se pot utiliza ca intrări de uz general și pot fi citite din RR0. Tranzițiile care apar pe aceste linii pot genera întreruperi dacă sunt validate întreruperile de stare/externe.

DTRA, DTRB (Data Terminal Ready) - terminal de date pregătit. Liniile sunt ieșiri de uz general, care urmăresc starea bitului DTR din registrul WR5.

DCDA, DCDB (Data Carrier Detect) - detecție purtătoare. Aceste linii sunt intrări trigger-Schmitt care, dacă se lucrează în modul cu autovalidare, servesc la validarea circuitelor de recepție. Altfel, pot fi folosite ca intrări de uz general, care pot să genereze întreruperi la orice tranziție dacă sunt validate întreruperile de stare/externe.

5.6.2. Funcționarea circuitului SIO-Z80

Cele două canale ale circuitului funcționează în paralel, complet independent unul față de celălalt și pot să asigure implementarea protocoalelor asincrone și sincrone de comunicație serială. Funcționarea circuitului este guvernată de conținutul registrelor de comandă. Registrele de stare

pot fi citite în orice moment de către unitatea centrală, dar numai unele comenzi și moduri pot fi modificate în timpul unei funcționări deja programate.

5.6.2.1. Dialogul cu unitatea centrală de procesare

Există trei posibilități de realizare a transferului de date, comenzi și stări între unitatea centrală și circuitul SIO-Z80:

- interogarea perifericelor (*polling*);
- utilizarea întreruperilor;
- transferul pe blocuri de octeți.

Tehnica interogării perifericelor necesită verificarea, prin program, a condițiilor de realizare a transferului. În acest scop, circuitul conține câte două registre de stare pentru fiecare canal, RR0 și RR1, care sunt actualizate permanent de către logica internă de comandă, în funcție de starea blocurilor de transmisie și de recepție.

Astfel, RR0 indică unității centrale momentele când circuitul are date recepționate sau are nevoie de date pentru transmisie. Acest registru trebuie consultat de fiecare dată, înainte de înscrierea unui nou octet în tamponul de transmisie, respectiv înainte de citirea unui caracter recepționat.

Registru RR1 conține condițiile de recepție și trebuie consultat după recepția fiecărui caracter, pentru a depista eventualele erori. În modul "polling" nu se utilizează întreruperile și deci ele trebuie invalidate la programarea registrului de comandă WR1.

Tehnica întreruperilor este favorizată de logica internă de gestionare a întreruperilor, care permite utilizarea circuitului în aplicații de timp real. În funcție de condițiile care le pot genera, cererile de întrerupere ale circuitului SIO-Z80 pot fi grupate în trei categorii, în ordinea descrescătoare a priorităților: *întreruperi de recepție*, de *transmisie* și *întreruperi de stare/externe*, canalul A fiind prioritar față de canalul B. Fiecare din aceste categorii de întreruperi poate fi validată separat prin program, la înscrierea registrului de comandă WR1.

La rândul lor, **întreruperile de recepție** pot fi de trei tipuri:

- la primul caracter recepționat;
- la fiecare caracter recepționat;
- datorate condițiilor speciale de recepție.

Primelor două tipuri le corespunde un același vector de întrerupere, numit în cele ce urmează *vectorul de întrerupere de recepție*. Odată validat unul din ele, în mod automat este validat și cel de-al treilea tip, căruia îi corespunde un al doilea vector de întrerupere: *vectorul de întrerupere de condiții speciale la recepție*. O astfel de întrerupere poate să fie generată, de exemplu, la apariția unei erori de recepție (paritate, încadrare, supraînscriere) sau la detectarea unui sfârșit de cadru - în protocolul SDLC.

Întreruperile la primul caracter recepționat se utilizează în cazul transferului pe blocuri de octeți. În acest caz, condițiile speciale de recepție - cu excepția erorii de paritate - pot genera întreruperi la recepția oricărui caracter al blocului de date, nu numai a primului caracter.

Întreruperile la fiecare caracter recepționat permit modificarea vectorului de întrerupere la detectarea unei erori de paritate, astfel încât și această eroare să poată genera o întrerupere de condiții speciale de recepție.

Validarea **întreruperilor de transmisie** va determina întreruperea unității centrale la golirea primului nivel al tamponului de transmisie (registru de transmisie date din fig.5.42), după ce, în prealabil, acesta a fost încărcat cu un octet de date. Acestor întreruperi le corespunde un al treilea vector: *vectorul de întrerupere de transmisie*.

Înteruperele de stare/externe au rolul de a semnaliza tranzițiile semnalelor de la intrările $\overline{\text{CTS}}$, $\overline{\text{DCD}}$ și $\overline{\text{SYNC}}$ (ultimul numai în modul sincron). Ele marchează apariția unor evenimente externe sau indică o schimbare în starea modemului sau a altor echipamente periferice. Acest tip de întrerupere are propriul său vector: *vectorul de întreruperi de stare/externe*.

Se observă existența a 4 vectori de întrerupere pentru fiecare canal, deci a 8 vectori de întrerupere pentru fiecare circuit SIO-Z80. Această structură de întreruperi este utilizată numai dacă bitul D_2 (starea afectează vectorul), din registrul de comandă WR1 al canalului B, este programat pe "1". Dacă D_2 este programat pe "0", se obține o altă variantă de tratare a întreruperilor, mult mai rudimentară și tocmai de aceea mai puțin utilizată în practică. În acest caz, structura sistemului de întreruperi se reduce la un singur vector de întrerupere, comun pentru ambele canale și pentru toate condițiile de întrerupere. Aceasta complică structura rutinei de întrerupere, care trebuie să identifice cu exactitate cauza care a produs întreruperea înainte de a trece la tratarea ei.

5.6.2.2. Posibilități de implementare a protocoalelor asincrone

Circuitul SIO-Z80 poate să implementeze, pe oricare din cele două canale, nivelul fizic al protocolului de comunicație serie asincron descris în §3.3.1. În continuare se va insista doar asupra unor aspecte particulare ale utilizării lui în comunicații asincrone de date și a unor facilități speciale de care dispune în acest scop.

Viteza de comunicație se poate selecta ca fiind 1/1, 1/16, 1/32 sau 1/64 din frecvența semnalelor de tact. Linia $\overline{\text{SYNC}}$ poate fi utilizată ca intrare pentru un semnal de tip indicator de apel (RI), generat de modem (vezi §3.2.1).

Transmisia asincronă poate începe numai după validarea transmițătorului (bitul $D_3=1$ în registrul de comandă WR5). Dacă a fost programat modul "autovalidare" (bitul $D_5=1$ în registrul de comandă WR3), atunci începerea transmisiei va fi condiționată, în mod suplimentar și de activarea intrării $\overline{\text{CTS}}$. Circuitul poate forța transmisia caracterului BREAK ("0" continuu pe linie) prin setarea bitului D_4 din registrul de comandă WR5. Revenirea la funcționarea normală se poate face fie prin resetarea bitului respectiv, fie prin inițializarea circuitului.

Recepția asincronă poate începe numai după validarea receptorului (bitul $D_0=1$ în registrul de comandă WR3). Dacă a fost programat modul "autovalidare", atunci începerea recepției datelor va fi, de asemenea, condiționată și de activarea liniei de intrare $\overline{\text{CTS}}$. Receptorul permite detecția biților de START falși, adică apariția, în starea de pauză - când linia RxD este în "1" - a unor stări "0" pe durata a mai puțin de $1/2$ din perioada corespunzătoare vitezei de comunicație.

Circuitele de recepție conțin o stivă de caractere recepționate, organizată pe trei niveluri și o stivă paralelă, cu indicatorii de eroare. Indicatorii de eroare pentru caracterul curent pot fi citiți din registrul cu condiții speciale de recepție, RR1. Indicatorii de eroare de paritate și de supraînscriere sunt cumulativi, adică nu pot fi șterși decât printr-o comandă de inițializare erori (vezi registrul de comandă WR0). Indicatorii sfârșit de cadru și eroare CRC încadrare din RR1 reflectă starea curentă a caracterului aflat în vârful stivei de date FIFO și nu sunt afectați de comanda de inițializare erori. Este posibil ca în octetul de stare, citit din RR1 după preluarea caracterului din vârful stivei de date, să fie incluse și erorile următorului caracter - în cazul în care acesta a fost recepționat. De aceea, pentru a păstra legătura dintre caracterul aflat în vârful stivei de date și cuvântul de stare din RR1, este necesar ca registrul de stare să fie citit înaintea datei din vârful stivei. Condiția este îndeplinită ușor la transferul prin întreruperi, când starea afectează vectorul (bitul $D_2=1$ în WR1), deoarece la apariția unei erori se generează automat o întrerupere de

Înteruperile de stare/externe au rolul de a semnaliza tranzițiile semnalelor de la intrările $\overline{\text{CTS}}$, $\overline{\text{DCD}}$ și $\overline{\text{SYNC}}$ (ultimul numai în modul sincron). Ele marchează apariția unor evenimente externe sau indică o schimbare în starea modemului sau a altor echipamente periferice. Acest tip de întrerupere are propriul său vector: *vectorul de întreruperi de stare/externe*.

Se observă existența a 4 vectori de întrerupere pentru fiecare canal, deci a 8 vectori de întrerupere pentru fiecare circuit SIO-Z80. Această structură de întreruperi este utilizată numai dacă bitul D_2 (starea afectează vectorul), din registrul de comandă WR1 al canalului B, este programat pe "1". Dacă D_2 este programat pe "0", se obține o altă variantă de tratare a întreruperilor, mult mai rudimentară și tocmai de aceea mai puțin utilizată în practică. În acest caz, structura sistemului de întreruperi se reduce la un singur vector de întrerupere, comun pentru ambele canale și pentru toate condițiile de întrerupere. Aceasta complică structura rutinei de întrerupere, care trebuie să identifice cu exactitate cauza care a produs întreruperea înainte de a trece la tratarea ei.

5.6.2.2. Posibilități de implementare a protocoalelor asincrone

Circuitul SIO-Z80 poate să implementeze, pe oricare din cele două canale, nivelul fizic al protocolului de comunicație serie asincron descris în §3.3.1. În continuare se va insista doar asupra unor aspecte particulare ale utilizării lui în comunicații asincrone de date și a unor facilități speciale de care dispune în acest scop.

Viteza de comunicație se poate selecta ca fiind 1/1, 1/16, 1/32 sau 1/64 din frecvența semnalelor de tact. Linia $\overline{\text{SYNC}}$ poate fi utilizată ca intrare pentru un semnal de tip indicator de apel (RI), generat de modem (vezi §3.2.1).

Transmisia asincronă poate începe numai după validarea transmițătorului (bitul $D_3=1$ în registrul de comandă WR5). Dacă a fost programat modul "autovalidare" (bitul $D_5=1$ în registrul de comandă WR3), atunci începerea transmisiei va fi condiționată, în mod suplimentar și de activarea intrării $\overline{\text{CTS}}$. Circuitul poate forța transmisia caracterului BREAK ("0" continuu pe linie) prin setarea bitului D_4 din registrul de comandă WR5. Revenirea la funcționarea normală se poate face fie prin resetarea bitului respectiv, fie prin inițializarea circuitului.

Recepția asincronă poate începe numai după validarea receptorului (bitul $D_0=1$ în registrul de comandă WR3). Dacă a fost programat modul "autovalidare", atunci începerea recepției datelor va fi, de asemenea, condiționată și de activarea liniei de intrare $\overline{\text{CTS}}$. Receptorul permite detecția biților de START falși, adică apariția, în starea de pauză - când linia RxD este în "1" - a unor stări "0" pe durata a mai puțin de $1/2$ din perioada corespunzătoare vitezei de comunicație.

Circuitele de recepție conțin o stivă de caractere recepționate, organizată pe trei niveluri și o stivă paralelă, cu indicatorii de eroare. Indicatorii de eroare pentru caracterul curent pot fi citiți din registrul cu condiții speciale de recepție, RR1. Indicatorii de eroare de paritate și de supraînscrere sunt cumulativi, adică nu pot fi șterși decât printr-o comandă de inițializare erori (vezi registrul de comandă WR0). Indicatorii sfârșit de cadru și eroare CRC/încadrare din RR1 reflectă starea curentă a caracterului aflat în vârful stivei de date FIFO și nu sunt afectați de comanda de inițializare erori. Este posibil ca în octetul de stare, citit din RR1 după preluarea caracterului din vârful stivei de date, să fie incluse și erorile următorului caracter - în cazul în care acesta a fost recepționat. De aceea, pentru a păstra legătura dintre caracterul aflat în vârful stivei de date și cuvântul de stare din RR1, este necesar ca registrul de stare să fie citit înaintea datei din vârful stivei. Condiția este îndeplinită ușor la transferul prin întreruperi, când starea afectează vectorul (bitul $D_2=1$ în WR1), deoarece la apariția unei erori se generează automat o întrerupere de

condiție specială de recepție. O excepție o constituie validarea întreruperii la recepția primului caracter, deoarece, în acest caz, o întrerupere generată de o condiție specială face ca starea, ca și caracterul, să fie păstrate până la o comandă de inițializare erori, deci ordinea de citire a caracterului recepționat și a informației de stare nu mai contează.

Când se lucrează în modul "polling", unitatea centrală trebuie să aștepte poziționarea indicatorului de caracter recepționat disponibil (bitul $D_0=1$ în RR0), înainte de a citi caracterul din vârful stivei de date. La transmisie, pentru a evita suprapunerea de caractere, trebuie să se aștepte poziționarea indicatorului tampon de transmisie gol (bitul $D_2=1$ în RR0) înainte de a înscrie un nou octet de date.

5.6.2.3. Posibilități de implementare a protocoalelor sincrone

Circuitul SIO-Z80 permite implementarea atât a protocoalelor sincrone orientate pe caracter (BCP), cât și a celor orientate pe bit (BOP) (v. §3.3.3).

După inițializarea circuitului și după o comandă de "validare-recepție" (bitul $D_0=1$ în WR3), receptorul trece în starea "așteptare-sincronizare" (hunt mode). Asamblarea caracterelor va începe numai după realizarea sincronizării. Viteza de transfer este întotdeauna egală cu frecvența semnalului de tact de transmisie $\overline{\text{TxC}}$, respectiv de recepție $\overline{\text{RxC}}$. La transmisie, schimbările de stare pe linia TxD se produc sincron cu frontul descrescător al semnalului de tact $\overline{\text{TxC}}$, iar la recepție starea liniei RxD este eșantionată pe frontul crescător al semnalului de tact $\overline{\text{RxC}}$. Asamblarea caracterelor va continua până la întâlnirea uneia din următoarele condiții:

- inițializarea circuitului (prin activarea intrării RESET sau printr-o comandă de "inițializare-canal" în registrul de comandă WR0);
- invalidarea recepției prin program (bitul $D_0=0$ în WR3) sau prin dezactivarea intrării $\overline{\text{DCD}}$ (în modul cu autovalidare);
- o nouă comandă de intrare în modul "așteptare-sincronizare" (bitul $D_4=1$ în WR3), care se poate aplica și la pierderea sincronismului.

Circuitul poate genera și verifica automat resturile polinomiale, folosind unul din polinoamele generatoare CRC-16 sau CCITT. Dacă nu mai sunt date de transmis, circuitul poate transmite caracterele CRC în mod automat, fără intervenția unității centrale, la sfârșitul blocului de date; deci se pot efectua transferuri de mare viteză, de tip DMA. Dacă nu mai sunt de transmis nici caractere CRC, circuitul transmite continuu, pe linie, caractere de sincronizare.

Implementarea protocoalelor BCP

În funcție de modul în care se realizează sincronizarea, protocoalele BCP pot fi:

- cu sincronizare internă - ieșirea $\overline{\text{SYNC}}$ este activată pe durata perioadei de ceas în care s-a obținut sincronizarea, care poate fi:
 - de tip MONOSYNC - asamblarea caracterelor începe la detecția unui singur caracter de sincronizare, programat în registrul de comandă WR7;
 - de tip BISYNC - asamblarea caracterelor începe la detecția celor două caractere de sincronizare, consecutive, programate în registrele de comandă WR6 și WR7.
- cu sincronizare externă - asamblarea caracterelor începe odată cu primul front pozitiv al semnalului de ceas $\overline{\text{RxC}}$ de după activarea intrării $\overline{\text{SYNC}}$, intrare care trebuie menținută pe "0" logic cel puțin trei perioade de ceas (CLK).

Detecția caracterelor de sincronizare nu generează cerere de întrerupere. Verificarea CRC se întârzie cu un caracter, astfel încât unitatea centrală poate excepta caracterele de control de la calculul CRC.

Transmisia sincronă de tip BCP

După inițializare, când transmisia este dezactivată, linia TxD este menținută în starea MARK ("1" logic). După selectarea modului și validarea transmisiei, circuitul transmite continuu, pe linie, caractere de sincronizare. Unitatea centrală poate lucra cu circuitul în modul cu interogare sau prin întreruperi. La rândul lor, întreruperile pot fi de două tipuri: de transmisie și de stare/externe.

Validarea întreruperilor de transmisie se face prin înscrierea, în registrul de comandă WR1, a unui cuvânt cu bitul $D_1=1$. De fiecare dată când tamponul de transmisie se golește, va fi generată o cerere de întrerupere de transmisie. În rutina de tratare a întreruperii de transmisie se poate transmite următorul caracter sau, dacă nu mai sunt caractere de transmis, se achită întreruperea printr-o comandă "inițializare-întrerupere-transmisie-în-așteptare" (în WR0).

Validarea întreruperilor de stare/externe se face prin înscrierea în registrul de comandă WR1 a unui cuvânt cu bitul $D_0=1$. O astfel de întrerupere poate să apară la începerea transmisiei caracterelor CRC, a celor de sincronizare sau la detectarea unor tranziții pe liniile \overline{CTS} , \overline{DCD} și \overline{SYNC} .

Dacă protocolul BCP utilizează controlul CRC, trebuie validată transmisia caracterelor CRC (bitul $D_0=1$ în registrul de comandă WR5), iar datele trebuie să fie transmise numai după o comandă "inițializare-generator-CRC-la-transmisie". După o inițializare externă sau internă, indicatorul "eroare-de-ritm-la-transmisie/EOM" (bitul D_6 din RR0) este pus pe "1". Cât timp acest indicator rămâne "1", chiar dacă nu mai sunt date de transmis, este inhibată transmiterea caracterelor CRC, permițându-se astfel circuitului să insereze automat caractere de sincronizare. Acest lucru este util în cazul în care microprocesorul întârzie să trimită următorul caracter al blocului BCP. Cel mai devreme după transmisia primului caracter al blocului BCP, dar de obicei imediat după înscrierea ultimului octet de date, acest indicator trebuie resetat, folosind o comandă de "inițializare-eroare-de-ritm-la-transmisie/EOM" ($D_7=D_6=1$ în WR0). Se permite astfel circuitului să insereze, la sfârșitul blocului de date, caracterele CRC. Imediat ce începe transmisia caracterelor CRC, indicatorul "eroare-de-ritm-la-transmisie/EOM" revine în "1" și se generează o întrerupere de stare/externă. În rutina de tratare, dacă nu mai sunt blocuri BCP de transmis (End Of Message), se re setează din nou indicatorul de eroare-de-ritm-la-transmisie/EOM. Altfel, după transmisia caracterelor CRC și la începerea transmisiei de caractere de sincronizare, când indicatorul "tampon-de-transmisie-gol" devine "1", se generează o întrerupere de transmisie, indicând posibilitatea de a începe transmisia următorului bloc BCP.

Sunt exceptate de la calculul CRC numai caracterele de sincronizare inserate automat, nu și cele existente în șirul de date. La programarea pe "1" a bitului "transmisie-BREAK" (bitul D_4 din WR5), linia TxD va fi forțată imediat în starea SPACE ("0" logic), indiferent de conținutul registrului de transmisie, ceea ce conduce la pierderea caracterelor în curs de transmisie. Invalidarea transmisiei ($D_3=0$ în WR5) în timpul emisiei unui caracter determină trecerea liniei TxD în starea MARK, dar numai după terminarea normală a transmisiei caracterului curent. Dacă există un caracter în tamponul de transmisie, acesta se păstrează.

Recepția sincronă de tip BCP

După inițializarea canalului (WR0), programarea receptorului necesită, în ordine, programarea modului (WR4), încărcarea caracterelor de sincronizare (WR6, WR7) și validarea recepției (WR3). Receptorul trece în starea "așteptare-sincronizare", iar după intrarea în sincronism trece la asamblarea caracterelor ce sosesc pe linia RxD. Unitatea centrală poate să lucreze cu receptorul prin interogare sau prin întreruperi: "întreruperi-la-primul-caracter-recepționat" sau "întreruperi-la-fiecare-caracter-recepționat".

Primul tip de întreruperi se poate utiliza pentru a lansa:

- o buclă de testare prin program - sincronizare prin interogare între unitatea centrală și circuitul Z80 SIO;
- o instrucțiune de I/E de transfer pe bloc (INIR, INDR) pentru restul caracterelor blocului BCP, caz în care se folosește linia $\overline{W/RDY}$ (având funcția WAIT) pentru sincronizarea unității centrale cu circuitul SIO-Z80;
- un transfer DMA, utilizând un controler specializat, caz în care se folosește linia $\overline{W/RDY}$ (având funcția READY) pentru sincronizarea circuitului SIO-Z80 cu controlerul DMA.

Circuitul Z80 SIO va genera o cerere de întrerupere numai la recepția primului caracter al blocului de date și, apoi, numai în caz de eroare (cu excepția erorii de paritate). După recepția întregului bloc de date se achită întreruperea cu o comandă "inițializare-întrerupere-la-primul-caracter-recepționat" (WR0).

În cel de-al doilea caz, se generează o întrerupere ori de câte ori există un caracter recepționat în vârful stivei FIFO, gata de a fi preluat de unitatea centrală. Condițiile speciale de recepție (inclusiv eroarea de paritate) pot genera un vector de întrerupere propriu, dacă în prealabil a fost selectat modul "starea-afectează-vectorul" (WR1).

Calculul restului CRC începe cu o întârziere de 8 perioade de tact \overline{RxC} față de momentul în care caracterul curent a fost transferat din registrul cu deplasare în stiva FIFO de recepție. Acest lucru face posibilă exceptarea caracterelor de control sau a secvențelor transparente de la calculul CRC.

Implementarea protocoalelor BOP

Protocoalele BOP, cum sunt SDLC sau HDLC, se implementează ușor folosind circuitul SIO-Z80, care asigură transmiterea automată a delimitatorului, inserarea și eliminarea biților "0", generarea și controlul CRC. Receptorul se sincronizează în mod automat pe delimitatorul de început al unui cadru, când generează un semnal de sincronizare pe ieșirea \overline{SYNC} sau, dacă a fost programat în acest sens, o cerere de întrerupere. Circuitul mai poate fi programat pentru a căuta cadre adresate unui anumit sistem sau tuturor sistemelor din rețea (cadre difuzate, cu adresa FFh), cu ignorarea tuturor cadrelor care nu îndeplinesc aceste condiții.

Transmisia sincronă de tip BOP

După selectarea modului (WR4) și validarea transmisiei (WR5), circuitul emite continuu pe linie delimitatoare (caractere 01111110b). Înainte de transmisia unui bloc de date, trebuie comandată "inițializarea-generatorului-CRC-la-transmisie" (WR0).

Indicatorul "eroare-de-ritm-la-transmisie/EOM" trebuie utilizat altfel decât în cazul transmisiei BCP, întrucât delimitatoarele, inserate automat atunci când unitatea centrală nu trimite la timp caracterele în tamponul de transmisie, ar fi interpretate la recepție, în mod eronat, drept sfârșit de cadru.

Astfel, indicatorul "eroare-de-ritm-la-transmisie/EOM" este resetat chiar la începutul transmisiei unui cadru. În cazul în care unitatea centrală nu trimite la timp caracterele în tamponul de transmisie, se transmit automat caracterele CRC și se generează o întrerupere de stare/externă caracter de date dintr-un cadru, tratarea acestei întreruperi trebuie să conțină o comandă "transmisie-ABORT" (WR0), pentru ca receptorul să renunțe la cadrul curent și să aștepte retransmiterea acestuia. Caracterele aflate în curs de transmisie sau în tamponul de transmisie se vor pierde. Apoi, înainte de revenirea din întrerupere, trebuie să se reseteze din nou indicatorul "eroare-de-ritm-la-transmisie/EOM". Dacă întreruperea a apărut la sfârșitul blocului de date, nu se va comanda generarea secvenței ABORT, ci doar resetarea indicatorului respectiv, și asta numai

dacă a fost ultimul cadru al unui mesaj (End Of Message). Dacă nu a fost ultimul cadru și indicatorul nu se inițializează, la terminarea transmisiei caracterelor CRC se inserează în mod automat delimitatorul de sfârșit de cadru și se generează o întrerupere de transmisie, ce poate fi folosită pentru începerea transmisiei următorului cadru.

Inserarea de biți "0" se face automat de către SIO-Z80, cu excepția delimitatoarelor (6 biți succesivi "1") și a secvențelor ABORT (mai mult de 7 biți "1" succesivi).

Se pot transmite caractere având 1÷8 biți de date, numărul de biți/caracter putând fi schimbat chiar și în cursul transmisiei unui cadru. Pentru sincronizarea cu unitatea centrală se pot folosi aceleași metode ca și la transmisia sincronă de tip BCP.

Recepția sincronă de tip BOP

După inițializarea canalului (WR0), selecția modului de lucru (WR4), înscrierea adresei (WR7) și validarea recepției (WR3), receptorul intră în modul "așteptare-sincronizare". Recepția datelor poate începe la detectarea primului caracter care urmează unui delimitator sau, dacă se lucrează în modul "căutare-adresă" (WR3), doar la detectarea adresei programate, respectiv a adresei globale. Atunci când nu se lucrează în modul "căutare-adresă" sau atunci când câmpul de adresă are doi octeți (HDLC), este posibil să se renunțe la recepția completă a cadrului curent printr-o comandă de reintrare în modul "așteptare-sincronizare" (bitul $D_4=1$ în WR3).

Transferul de date se poate desfășura prin interogare sau prin întreruperi. De exemplu, prin selectarea modului "întrerupere-la-primul-caracter-recepționat" (WR1) se generează o cerere de întrerupere la începutul recepției fiecărui cadru. Dacă nu se cunoaște dinainte numărul de caractere al cadrului, se poate ieși din bucla de recepție pe "condiția-de-recepție-specială" de "detectare-sfârșit-de-cadru", care poate genera o cerere de întrerupere. Atunci când numărul de biți din câmpul de date nu este un multiplu întreg al lungimii caracterelor, trebuie utilizată valoarea câmpului "cod-de-rest-I-la-recepție", din registrul de stare RR1, pentru a determina numărul efectiv de biți de informație din ultimele două caractere recepționate. Apoi, se reactivează "întreruperea-la-primul-caracter-recepționat" cu o comandă corespunzătoare de inițializare (WR0).

Biții "0" inserați în mod automat la transmisie pentru păstrarea transparenței cadrelor în raport cu secvența delimitator sunt rejectați în mod automat la recepție. Detectarea unei secvențe ABORT (mai mult de 7 biți "1" succesivi) determină setarea bistabilului BREAK/ABORT din RR0 și generarea, dacă a fost validată, a unei întreruperi de stare/externe. În rutina de tratare se iau măsuri pentru renunțarea la recepția cadrului curent, reintrarea în modul "așteptare-sincronizare" cu sau fără căutare adresă (WR3), după care se reactivează, cu comenzi succesive de inițializare (WR0), "întreruperea-la-primul-caracter-recepționat" și întreruperea externă/de stare.

Validarea verificării CRC se face automat, de către delimitatorul de început și continuă până la detectarea delimitatorului de sfârșit de cadru. Rezultatul verificării CRC este dat de indicatorul "eroare-de-CRC/incadrare" din RR1, fiind "0" pentru un cadru corect recepționat. Verificarea de paritate poate fi implementată doar prin program, numai pentru câmpul de date al unui cadru și necesită realizarea unei legături semiduplex (cu confirmare), deoarece în acest mod nu se pot utiliza circuitele de generare/control automat al parității.

5.6.3. Programarea circuitului SIO-Z80

Programarea circuitului SIO-Z80 se face prin încărcarea registrelor de comandă ale celor două canale, cu comenzi și parametri specifici modului de funcționare dorit și prin consultarea registrelor de stare, care conțin informații despre starea curentă a fiecărui canal. O parte din registrele de comandă pot fi modificate în timpul funcționării, prin comenzi sau parametri de lucru specifici modului de operare programat inițial, fără a fi necesară reinițializarea canalelor. Cele trei variante constructive (SIO-Z80/0/1/2) sunt identice din punctul de vedere al programării.

5.6.3.1. Registrele de comandă

Canalul A conține un număr de 7 registre de comandă (WR0÷WR7, mai puțin WR2), iar canalul B 8 registre de comandă (WR0÷WR7). Registrul WR0 se programează prin înscrierea unui singur octet pe adresa portului de comandă al canalului corespunzător, iar celelalte registre de comandă au nevoie fiecare de 2 octeți, care se trimit în ordine pe aceeași adresă: primul octet este destinat registrului WR0 și are rolul de a selecta registrul căruia îi este destinat cel de-al doilea octet. La inițializare este selectat în mod implicit registrul WR0, de aceea pentru el au fost rezervate principalele comenzi de lucru.

Registrul de comandă WR0

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
Cod inițializare CRC		Cod comandă			Cod selecție registru următor		

Câmpul "cod-selecție-registru-următor" reprezintă numărul registrului selectat pentru a primi următorul octet, exprimat în binar pe 3 biți: D₂, D₁ și D₀.

Câmpul "cod-comandă" codifică cele 8 comenzi care se pot transmite unui canal:

Cod c-dă			Denumirea comenzii	Observații
D ₇	D ₆	D ₅		
0	0	0	Comandă neoperantă	Nu are nici un efect; se utilizează la selecția registrelor WR1÷WR7.
0	0	1	Transmisie ABORT (modul SDLC)	Este utilizată numai în protocoalele BOP, pentru a iniția transmiterea unei secvențe ABORT (>7 biți "1" consecutivi).
0	1	0	Inițializare întreruperi externe/de stare	Determină re poziționarea biților de stare din RR0 după generarea unei întreruperi externe/de stare, care blocase acești biți în vederea citirii lor de către unitatea centrală.
0	1	1	Inițializare canal	Realizează aceeași inițializare ca și resetarea HW, dar numai pentru un canal (dacă e transmisă pe canalul A, inițializează și logica de tratare a întreruperilor).
1	0	0	Inițializare întrerupere la primul caracter recepționat	Reactivează acest tip de întrerupere, pregătind recepția următorului bloc de date.
1	0	1	Inițializare întrerupere transmisie în așteptare	Previne generarea unor noi întreruperi de transmisie, după golirea tamponului de transmisie, până la înscrierea în acesta a unui nou caracter de către unitatea centrală.
1	1	0	Inițializare erori	Șterge eventualele erori de paritate și/sau depășire, memorate în RR1 de la precedenta inițializare.
1	1	1	Revenire din întrerupere (doar canalul A)	Trebuie trimisă numai canalului A, fiind interpretată de către circuit ca o comandă de achitare a întreruperii în curs (ca și o instrucțiune RETI); comanda permite folosirea priorităților și în sistemele care nu se bazează pe microprocesorul Z80.

Câmpul "cod-inițializare-CRC" are semnificația din tabelul următor:

D ₇	D ₆	Denumirea comenzii	Observații
D ₇	D ₆		
0	0	Comandă neoperantă	Nu are nici un efect; se utilizează la selecția registrelor WR1÷WR7.
0	1	Inițializare verificare CRC la recepție	Determină inițializarea registrului de deplasare utilizat pentru verificarea CRC, după o eroare de CRC/incadrare.
1	0	Inițializare generare CRC la transmisie	Determină inițializarea registrului de deplasare utilizat pentru generarea CRC, înainte de transmiterea unui nou bloc de date.
1	1	Inițializare eroare de ritm/EOM	Determină resetarea bistabilului cu același nume din RR0.

Partea a II-a - Sisteme cu microprocesoare de 8 biți

Registrul de comandă WR1

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
Validare WAIT/READY	WAIT/READY	W / RDY la recepție/transmisie	Mod de întrerupere la recepție		Starea afec-tează vecto-rul (canal B)	Validare întreruperi la transmisie	Validare întreruperi de stare/externe

D₀ = 1 - activează generarea întreruperilor ca urmare a unor tranziții pe liniile $\overline{\text{CTS}}$, $\overline{\text{DCD}}$ și $\overline{\text{SYNC}}$, a întâlnirii unei secvențe BREAK ("0" continuu pe linie pe durata a mai mult de un caracter) sau la începutul transmisiei caracterelor CRC ori SYNC;

D₁ = 1 - permite generarea unei întreruperi ori de câte ori tamponul de transmisie devine gol;

D₂ = 1 - stabilește, numai dacă se înscrie în canalul B, modificarea vectorului de întrerupere înscris de utilizator în registrul WR2 al canalului B, conform tabelului de mai jos:

V ₃	V ₂	V ₁	Semnificația modificării	Observații
0	0	0	Tampon transmisie gol	Canalul B
0	0	1	Schimbare externă/de stare	
0	1	0	Caracter recepționat disponibil	
0	1	1	Condiție de recepție specială	
1	0	0	Tampon transmisie gol	Canalul A
1	0	1	Schimbare externă/de stare	
1	1	0	Caracter recepționat disponibil	
1	1	1	Condiție de recepție specială	

Dacă D₂=0, vectorul trimis de SIO-Z80 este chiar cel înscris prin program în registrul WR2.

D₄ și D₃ stabilesc modul de întrerupere la recepție, conform tabelului următor:

D ₄	D ₃	Mod de întrerupere la recepție
0	0	Întrerupere de recepție invalidată.
0	1	Întrerupere la primul caracter recepționat.
1	0	Întrerupere la fiecare caracter recepționat. Paritatea afectează vectorul.
1	1	Întrerupere la fiecare caracter recepționat. Paritatea nu afectează vectorul.

D₅ selectează condiția în care ieșirea $\overline{\text{W / RDY}}$ va fi activată: când tamponul de recepție este gol (D₅=1) sau când tamponul de transmisie este plin (D₅=0).

D₆ selectează funcția îndeplinită de linia de ieșire $\overline{\text{W / RDY}}$: când această linie comandă o intrare de tip $\overline{\text{WAIT}}$ a unui microprocesor, D₆=1; când comandă o intrare de tip READY a unui controler DMA, D₆=0. Funcția READY este activă în orice moment, nefiind condiționată de selectarea prealabilă a circuitului, în timp ce funcția WAIT este activă numai atunci când se încearcă citirea din tamponul de recepție care este gol, sau înscrierea în tamponul de transmisie care este plin (în funcție de D₅).

D₇ = 1 - validează funcționarea liniei $\overline{\text{W / RDY}}$ într-unul din cele două moduri descrise mai sus, iar D₇=0 forțează linia $\overline{\text{W / RDY}}$ în "1", dacă s-a selectat funcția READY, respectiv o trece în starea de înaltă impedanță - pentru funcția WAIT.

Registrul de comandă WR2

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
V7	V6	V5	V4	V3	V2	V1	0
Nu se modifică				Se pot modifica numai dacă D ₂ =1 în WR1 (canal B)			Nu se modifică

Registrul WR2 nu există decât pentru canalul B. El conține vectorul de întrerupere folosit de ambele canale, care va fi depus de circuit pe magistrala de date în ciclul de achitare a întreruperii, indiferent de cauza care a produs întreruperea - dacă bitul $D_2=0$ în WR1. Dacă bitul $D_2=1$ în WR1, circuitul modifică parțial acest vector, mai exact biții V_3 , V_2 și V_1 , conform tabelului prezentat la descrierea registrului de comandă WR1.

Registrul de comandă WR3

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Lungime caracter la recepție		Auto-validare	Așteptare-sincronizare	Validare CRC la recepție	Căutare adresă	Invalidare încărcare caracter SYNC	Validare recepție

$D_0 = 1$ - validează începerea recepției;

$D_1 = 1$ - previne încărcarea în coada FIFO a caracterelor SYNC recepționate;

$D_2 = 1$ - în modul sincron de tip BOP, va rejecta toate cadrele care au o adresă diferită de cea programată sau de adresa globală (FFh);

$D_3 = 1$ - determină inițierea sau reluarea calculului CRC, începând cu ultimul caracter transferat din registrul cu deplasare în stiva FIFO, fără a se ține cont de caracterele precedente;

$D_4 = 1$ - comandă reîntrarea în modul de așteptare a secvenței de sincronizare;

$D_5 = 1$ - determină ca intrările \overline{DCD} și \overline{CTS} să fie folosite pentru validarea circuitelor de recepție și respectiv de transmisie;

D_7 și D_6 - stabilesc lungimea caracterelor asamblate la recepție, conform tabelului următor:

D_7	D_6	Număr de biți/caracter
0	0	5
0	1	6
1	0	7
1	1	8

Registrul de comandă WR4

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Viteza de lucru		Moduri de sincronizare		Număr biți de STOP		Tip paritate	Validare paritate

$D_0 = 1$ - determină, la transmisie, adăugarea la numărul de biți specificat de D_7 și D_6 din WR3 a bitului de paritate, iar la recepție - verificarea parității caracterului recepționat.

$D_1 = 1$ - stabilește utilizarea parității pare (bitul de paritate este egal cu suma modulo 2 a biților de date ai caracterului); $D_1=0$ stabilește paritatea impară (bitul de paritate este egal cu suma modulo 2 complementată a biților de date ai caracterului).

D_3 și D_2 - precizează numărul biților de STOP adăugați la transmisia asincronă:

D_3	D_2	Număr biți de STOP
0	0	Moduri sincrone
0	1	1
1	0	$1\frac{1}{2}$
1	1	2

D_5 și D_4 selectează modul de sincronizare, conform tabelului:

D_5	D_4	Modul de sincronizare
0	0	Secvență de sincronizare de 8 biți, transmisă prin program.
0	1	Secvență de sincronizare de 16 biți, transmisă prin program.
1	0	Mod SDLC (secvența de sincronizare 0111110b).
1	1	Sincronizare externă, prin semnal.

D_7 și D_6 - specifică factorul de divizare a frecvenței semnalelor de tact de transmisie/recepție

$\overline{TxC}/\overline{RxC}$, care intervine în calculul vitezei de comunicație. Modurile sincrone impun alegerea unui factor de divizare 1, în timp ce modurile asincrone pot lucra cu orice factor de divizare dintre cei precizați în tabelul alăturat.

D7	D6	Factor de divizare
0	0	1
0	1	16
1	0	32
1	1	64

Trebuie menționat că, în toate modurile de lucru, semnalul de tact CLK trebuie să aibă o frecvență de cel puțin 5 ori mai mare decât viteza de comunicație. În plus, pentru un factor de divizare 1, trebuie să se asigure o sincronizare la nivel de bit printr-un semnal de tact extern, comun pentru transmițător și receptor.

Registrul de comandă WR5

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
DTR	Lungime caracter la transmisie		Transmisie BREAK	Validare transmisie	CRC-16 /CCITT	RTS	Validare CRC la transmisie

$D_0 = 1$ - permite calculul și transmisia caracterelor CRC atunci când nu mai sunt date de transmis.

$D_1 = 1$ - forțează activarea liniei de ieșire \overline{RTS} . Dacă $D_1=0$, linia \overline{RTS} se dezactivează, dar numai după golirea tamponului de transmisie.

D_2 - selectează polinomul generator: $D_2=1$ pentru CRC-16 ($X^{16} + X^{15} + X^2 + 1$), respectiv $D_2=0$ pentru CCITT ($X^{16} + X^{12} + X^5 + 1$).

$D_3 = 0$ - determină menținerea liniei \overline{TxD} în starea MARK. Transmisia poate începe numai după poziționarea acestui bit pe "1". Repoziționarea lui pe "0" va încheia transmisia după terminarea caracterului curent sau imediat, dacă acesta este un caracter CRC.

$D_4 = 1$ - forțează imediat linia \overline{TxD} în starea SPACE.

D_6 și D_5 - precizează numărul de biți dintr-un octet care se transmit efectiv pe linie, conform tabelului următor:

D6	D5	Număr de biți/caracter
0	0	5 sau mai puțini
0	1	7
1	0	6
1	1	8

Biții de date (D) se transmit începând cu cel mai puțin semnificativ (D_0). Dacă se transmit 5 biți sau mai puțini, octetul înscris în tamponul de transmisie stabilește numărul exact de biți care se transmit pe linie, prin D_7 , D_6 , D_5 și D_4 , conform următorului tabel:

D7	D6	D5	D4	D3	D2	D1	D0	Număr de biți/caracter
1	1	1	1	0	0	0	D	1
1	1	1	0	0	0	D	D	2
1	1	0	0	0	D	D	D	3
1	0	0	0	D	D	D	D	4
0	0	0	D	D	D	D	D	5

$D_7 = 1$ - forțează activarea liniei de ieșire \overline{DTR} .

Registrul de comandă WR6

D7	D6	D5	D4	D3	D2	D1	D0
SYNC ₇	SYNC ₆	SYNC ₅	SYNC ₄	SYNC ₃	SYNC ₂	SYNC ₁	SYNC ₀
ADDR ₇	ADDR ₆	ADDR ₅	ADDR ₄	ADDR ₃	ADDR ₂	ADDR ₁	ADDR ₀

Registrul WR6 poate conține:

- cei mai puțin semnificativi 8 biți ai unei secvențe de sincronizare de 16 biți (BISYNC),
- o adresă (dacă este validat modul căutare adresă în cazul protocoalelor BOP) sau
- caracterul de sincronizare de 8 biți (MONOSYNC).

Registrul de comandă WR7

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
SYNC15	SYNC14	SYNC13	SYNC12	SYNC11	SYNC10	SYNC9	SYNC8
0	1	1	1	1	1	1	0

Registrul WR7 poate conține:

- cei mai semnificativi 8 biți ai unei secvențe de sincronizare de 16 biți (BISYNC) sau
- delimitatorul (FLAG-ul) specific protocolului BOP (01111110b în modul SDLC).

Obs.: Registrele de comandă WR6 și WR7 nu se utilizează în modul de sincronizare externă.

5.6.3.2. Registrele de stare

Circuitul SIO-Z80 dispune de 2 registre de stare pentru canalul A (RR0 și RR1), respectiv 3 pentru canalul B (RR0, RR1 și RR2). Citirea unui registru de stare trebuie precedată de înscrierea adresei acestuia (0, 1 sau 2) în registrul de comandă WR0.

Registrul de stare RR0

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
Detectare BREAK/ ABORT	Eroare de ritm la transmisie/ EOM	CTS	SYNC/ așteptare sincroni- zare	DCD	Tampon transmisie gol	Înterupere în așteptare (numai canalul A)	Caracter recepționat disponibil

D₀ = 1 - atunci când în stiva FIFO se află cel puțin un caracter recepționat.

D₁ - este accesibil numai pe canalul A (pe canalul B este întotdeauna "0") și este "1" dacă există o cerere de întrerupere la nivelul întregului circuit (neacceptată încă sau în curs de tratare, dar neachitată).

D₂ - trece în "1" ori de câte ori tamponul de transmisie devine gol, cu excepția transmisiei caracterelor CRC, în modurile sincrone.

D₃ - reflectă starea intrării $\overline{\text{DCD}}$ (D₃=1 dacă $\overline{\text{DCD}}$ =0) din momentul ultimei schimbări a unui indicator de stare/extern (DCD, CTS, SYNC/așteptare sincronizare, detectare BREAK/ABORT, eroare de ritm la transmisie/EOM).

D₄ - în modul asincron, indică starea intrării $\overline{\text{SYNC}}$. În modul sincron, este pus automat pe "1" la intrarea în modul "așteptare-sincronizare" (D₄=1 în WR3) și devine "0" în momentul obținerii sincronizării.

D₅ - reflectă starea intrării $\overline{\text{CTS}}$ (D₅=1 dacă $\overline{\text{CTS}}$ =0).

D₆ - este poziționat în "1" după inițializare. Atât timp cât este în "1", dacă după obținerea sincronizării unitatea centrală întârzie să trimită date în tamponul de transmisie, circuitul inserează automat caractere de sincronizare. Dacă acest indicator este resetat printr-o comandă de inițializare corespunzătoare (D₇=D₆=1 în WR0), atunci când unitatea centrală nu mai trimite caractere în tamponul de transmisie, circuitul începe transmisia caracterelor CRC, poziționează indicatorul "eroare-de-ritm-la-transmisie/EOM" în "1", păstrează indicatorul "tampon-de-transmisie-gol" (D₂ în RR0) pe "0" și generează o întrerupere de stare/externă, de eroare de ritm la transmisie (transmisie sub viteza normală). Ea poate fi tratată în mod specific sau poate fi ignorată, dacă s-a ajuns la sfârșitul unui bloc de date.

Partea a II-a - Sisteme cu microprocesoare de 8 biți

La terminarea transmisiei caracterelor CRC, se trece la inserarea de caractere de sincronizare (sau delimitatoare), indicatorul "tampon-de-transmisie-gol" devine "1" și, dacă indicatorul "eroare-de-ritm-la-transmisie/EOM" nu a fost resetat în rutina de întrerupere externă de stare, se generează o întrerupere de transmisie, indicând posibilitatea începerii transmisiei unui nou bloc de date. Dacă, indicatorul "eroare-de-ritm-la-transmisie/EOM" a fost resetat, această întrerupere nu mai apare (End of Message - EOM).

D₇ - se poziționează pe "1" la detectarea unei secvențe BREAK ("0" continuu pe linie pe durata a mai mult de un caracter) în modul asincron, sau la detectarea unei secvențe ABORT (mai mult de 7 biți "1" succesivi, în modul SDLC). Schimbarea stării acestui bit poate genera o întrerupere de stare/externă. După inițializarea acestor întreruperi (WR0), bitul revine în "0" la terminarea secvenței BREAK/ABORT.

Registrul de stare RR1

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
Sfârșit de cadru (SDLC)	Eroare de CRC/încadrare	Eroare depășire recepție	Eroare paritate	Codul de rest I la recepție			Toate caracterele au fost transmise

D₀ - devine "1" în modul asincron, la golirea completă a tamponului de transmisie. Modificarea lui nu generează cerere de întrerupere. În modul sincron, **D₀** este menținut în "1".

D₃, D₂ și D₁ - permit determinarea numărului de biți de informație (I) din ultimii doi octeți ai unui cadru BOP, în cazul în care numărul total de biți al acestuia nu este multiplu de numărul de biți/caracter programat la recepție (mod SDLC). Spre exemplu, dacă numărul de biți/caracter este 8, acest câmp poate fi folosit conform tabelului următor:

D ₃	D ₂	D ₁	Biți I în ultimul octet	Biți I în penultimul octet
1	0	0	0	3
0	1	0	0	4
1	1	0	0	5
0	0	1	0	6
1	0	1	0	7
0	1	1	0	8
1	1	1	1	8
0	0	0	2	8

Biții de informație sunt aliniați la dreapta în cadrul unui octet (cei mai puțin semnificativi). Pentru alte valori ale numărului de biți/caracter se pot construi tabele similare, pornind de la tabelul alăturat, în care sunt date codurile pentru cazul când limita ultimului caracter asamblat coincide cu limita câmpului de informație, respectiv cu începutul caracterelor CRC.

Biți/caracter	D ₃	D ₂	D ₁
8	0	1	1
7	0	0	0
6	0	1	0
5	0	0	1

D₄ - va fi poziționat pe "1" pentru orice caracter recepționat a cărui paritate recalculată nu coincide cu bitul de paritate recepționat (cu condiția ca paritatea să fie validată). Odată poziționat, bitul se memorează, până la transmiterea unei comenzi de inițializare erori (WR0).

D₅ - indică faptul că s-au recepționat mai mult de 4 caractere fără ca unitatea centrală să citească vreunul, astfel încât caracterul aflat în vârful stivei s-a pierdut. Eroarea este atașată numai

caracterului care s-a încărcat peste cel anterior, dar, după citirea acestuia, eroarea se memorează până la trimiterea unei comenzi de inițializare erori.

$D_6 = 1$ - indică, în modul asincron, nerecepționarea corectă a unui bit de STOP. Această eroare se atașează numai caracterului afectat. În modul sincron, bitul D_6 indică rezultatul verificării CRC a ultimului bloc de date recepționat.

D_7 - indică, în modul SDLC, detectarea unui delimitator de sfârșit de cadru, moment în care devin valizi biții de eroare de încadrare/CRC și cod rest I.

Registrul de stare RR2

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0
La fel ca biții din WR2				Modificați dacă $D_2=1$ în WR1			0

Acest registru poate fi citit numai pe canalul B. Dacă bitul "starea-afectează-vectorul" din WR1 a fost programat pe "1", registrul indică vectorul de întrerupere modificat (conform tabelului prezentat la descrierea registrului de comandă WR1), care corespunde condiției de întrerupere cu cea mai mare prioritate, activă la momentul citirii. Dacă nu este activă nici o condiție de întrerupere, $V_3 V_2 V_1 = 011$.

Exemplu: Se consideră un circuit SIO-Z80, selectat cu linia de adresă A_7 ($\overline{CE} = \overline{A_7}$) și având liniile C/\overline{D} și B/\overline{A} conectate la liniile de adresă A_0 și respectiv A_1 . Secvențele de program prezentate în continuare urmează după o inițializare hardware și programează canalul B să lucreze în modul asincron, iar canalul A în modul sincron, conform protocolului SDLC.

SIADAT	EQU	80h	; Adresa portului de date al canalului A.
SIACOM	EQU	81h	; Adresa portului de comandă/stare al canalului A.
SIBDAT	EQU	82h	; Adresa portului de date al canalului B.
SIBCOM	EQU	83h	; Adresa portului de comandă/stare al canalului B.
adresa	EQU	0	; Adresa canalului serial A (conform protocolului SDLC).
.....			
LD	A,HIGH(tabint)		
LD	I,A		; Încarcă în I octetul superior al adresei tabelului de întreruperi.
IM	2		; Stabilește modul 2 de întreruperi mascabile.
CALL	inisia		; Apelează rutina de inițializare a SIO-Z80.
EI			; Validează întreruperile.
.....			
inisia:			; Rutina de inițializare SIO-Z80.
; Programare canal B în modul asincron			
.....			
LD	A,18h		; Comandă inițializare canal B (WR0B).
OUT	(SIBCOM),A		
LD	A,02h		; Selectează WR2B.
OUT	(SIBCOM)		
LD	A,LOW(tabint)		; Înscrie vectorul de întreruperi în WR2B.
OUT	(SIBCOM),A		
LD	A,04h		; Selectează WR4B.
OUT	(SIBCOM),A		
LD	A,47h		; Factor de divizare 16, 1 bit de STOP, paritate pară.
LD	(SIBCOM),A		
LD	A,05h		; Selectează WR5B.
OUT	(SIBCOM),A		
LD	A,0AAh		; Activează DTR și RTS, 7 biți/caracter la transmisie, validare
			; transmisie.
OUT	(SIBCOM),A		

Partea a II-a - Sisteme cu microprocesoare de 8 biți

```

LD      A,03h      ; Selectează WR3B.
OUT     (SIBCOM),A
LD      A,0A1h     ; 7 biți/caracter la recepție, auto-validare, validare recepție.
OUT     (SIBCOM),A
LD      A,01h     ; Selectează WR1B.
OUT     (SIBCOM),A
LD      A,17h     ; Întrerupere-la-fiecare-caracter-recepționat (paritatea
OUT     (SIBCOM),A ; afectează vectorul), starea-afectează-vectorul, validare
                  ; întreruperi-la-transmisie și întreruperi de stare/externe.

```

; Programare canal A în modul sincron, conform protocolului SDLC.

```

LD      A,18h     ; Comandă inițializare canal A (WR0A).
OUT     (SIACOM),A
LD      A,04h     ; Selectează WR4A.
OUT     (SIACOM),A
LD      A,20h     ; Mod SDLC, fără paritate.
LD      (SIACOM),A
LD      A,46h     ; Selectează registrul WR6A, inițializare-verificare-CRC-la-
                  ; recepție.

OUT     (SIACOM),A
LD      A,adresa  ; Încărcare adresă mesaj SDLC.
OUT     (SIACOM),A
LD      A,87h     ; Selectează WR7A, inițializare-generare-CRC-la-transmisie.
OUT     (SIACOM),A
LD      A,7Eh     ; Încărcare delimitator SDLC (01111110b).
OUT     (SIACOM),A
LD      A,01h     ; Selectează WR1A.
OUT     (SIACOM),A
LD      A,13h     ; Întrerupere-la-fiecare-caracter-recepționat (paritatea
OUT     (SIACOM),A ; afectează vectorul), validare întreruperi-la-transmisie și
                  ; întreruperi de stare/externe.

LD      A,15h     ; Selectează WR5A, inițializare întreruperi de stare/externe.
OUT     (SIACOM),A
LD      A,0E9h    ; Activează DTR, 8 biți/caracter la transmisie, validare.
OUT     (SIACOM),A ; transmisie, validare-CRC-la-transmisie, CCIT.
LD      A,03h     ; Selectează WR3A.
OUT     (SIACOM),A
LD      A,0EDh    ; 8 biți/caracter la recepție, auto-validare, validare-CRC-la-
OUT     (SIACOM),A ; recepție, căutare-adresă, validare-recepție.
RET

```

.....
tabint: ; Aliniată la o adresă multiplu de 16 (2 canale × 4 adrese × 2 octeți)

```

DW      transb    ; Adresa rutinei de tratare a întreruperii de transmisie canal B.
DW      extstb    ; Adresa rutinei de întrerupere externă/de stare canal B.
DW      recepb    ; Adresa rutinei de tratare a întreruperii de recepție canal B.
DW      recspb    ; Adresa rutinei de întrerupere de condiție specială de recepție canal B.
DW      transa    ; Similar, pentru canalul A.
DW      extsta
DW      recepa
DW      recspa

```

Rutina de inițializare, Inlsio, poate fi mult mai avantajos implementată cu instrucțiuni de ieșire pe blocuri. Varianta prezentată mai jos asigură aceeași funcționare a circuitului, întrucât cele

două şiruri de octeţi (tabsib şi tabsia) sunt formate din octeţii de înscris în cele două canale, folosiţi şi în varianta anterioară.

; Rutina de iniţializare a SIO-Z80, folosind instrucţiuni ieşire pe blocuri.
 inisio:

LD	A,LOW(tabint)	
LD	(tabsib+2),A	; Înscris în tabel vectorul de întreruperi.
LD	HL,tabsib	; Încarcă în HL adresa tabelului cu parametri pentru canalul B.
LD	C,SIBCOM	; Încarcă în C adresa pe care se vor înscris parametrii.
LD	B,ingsib	; Încarcă în registrul B numărul de octeţi de înscris.
OTIR		; Programează canalul B.
LD	HL,tabsia	; Încarcă în HL adresa tabelului cu parametri pentru canalul A.
LD	C,SIACOM	; Încarcă în C adresa pe care se vor înscris parametrii.
LD	B,ingsia	; Încarcă în registrul B numărul de octeţi de înscris.
OTIR		; Programează canalul A.
RET		

.....
 tabsib: DB 18h,2,0,4,47h,5,0AAh,3,0A1h,1,17h

ingsib EQU \$-tabsib ; lungimea şirului de octeţi de înscris în canalul B

tabsia: DB 18h,4,20h,46h,adresa,87h,7Eh,1,17h,15h,0E9h,3,0EDh

ingsia EQU \$-tabsia ; lungimea şirului de octeţi de înscris în canalul A



SISTEME CU MICROPROCESOARE DE 8 BIȚI INTEGRATE (MICROCONTROLERE)

Microcontrolerele sunt sisteme cu microprocesoare, de dimensiuni reduse, integrate pe un singur cip. În afară de UCP, acestea conțin memorie de tip ROM/EPROM și RAM, precum și un număr limitat de porturi I/E paralele. De asemenea, posedă cel puțin un timer și cel puțin un canal de comunicație serială. Întrucât aceste entități formează principalele componente ale unui calculator, acestor microsisteme li s-a dat uneori și denumirea de microcalculatoare pe un singur cip ("single-chip microcomputer"). Spre deosebire de bine-cunoscutele minicalculatoare de buzunar ("handheld calculator"), puternic specializate, microcontrolerele se caracterizează prin facilități specifice comunicării și controlului mediului extern.

Principalele diferențe între sistemele clasice cu microprocesoare și microcontrolere se referă la:

- capacitatea microcontrolerelor de a prelucra informația în timp real, utilizând un sistem de întreruperi inclus;
- capabilitatea unui transfer I/E eficient;
- set de instrucțiuni orientat spre implementarea avantajoasă a algoritmilor de control;
- facilități pentru introducerea timpului în prelucrarea informației;
- includerea pe același cip a convertoarelor A/D și chiar a unor facilități D/A, sub forma unor circuite PWM (Pulse Width Modulation).

Toate facilitățile enumerate au condus la denumirea actuală de *microcontroler*, cu o dezvoltare impresionantă; practic, toate firmele producătoare de microprocesoare realizează în ultimii ani și astfel de dispozitive, apărând și noi producători.

Evident, existența pe același cip a principalelor componente specifice unui sistem cu microprocesor orientat spre control are mari avantaje, cum ar fi:

- optimizarea transferurilor între diferitele entități ale sistemului;
- reducerea consumului, prin folosirea de tehnologii VLSI avansate (HMOS, CHMOS, etc.);
- creșterea fiabilității, prin reducerea la maximum a conexiunilor externe;
- adaptarea economică la aplicație, cu folosirea de resurse externe minime (regimul "stand alone");
- simplificarea proiectării hardware și, în mare parte, și a programelor;
- cost și dimensiuni reduse.

Cele arătate mai sus fac din MCU (MicroController Unit) suportul ideal de implementare a conceptelor de automatizare de cost redus (low cost automation) și de conducere distribuită, deși inițial au fost produse pentru eficientizarea echipamentelor periferice și a bunurilor de larg consum.

Pe de altă parte, utilizarea microcontrolerelor se caracterizează printr-o serie de dezavantaje și limitări, cum ar fi:

- resurse interne limitate în raport cu sistemele cu microprocesoare;
- necesitatea expandării resurselor, în special a memoriei și a porturilor I/E;
- utilizarea multiplă a pinilor, ceea ce impune o foarte bună cunoaștere de către utilizator a regimurilor de lucru;

- orientarea producției de microcontrolere pe familii, cu componente specializate (de ex. cu ROM/EPROM pe cip sau fără ROM, cu sau fără convertor A/D, ș.a.), ceea ce înseamnă o bună cunoaștere a caracteristicilor diferitelor familii de microcontrolere realizate de un producător.

Aspectele evidențiate mai sus se constituie în tot atâtea probleme pe care în mod obișnuit trebuie să le rezolve un proiectant de sistem cu microcontrolere, și anume:

a) la nivel local, să aleagă cel mai avantajos MCU pentru o implementare optimă a aplicației, atât din punct de vedere al resurselor HW (ideal, funcționarea "stand alone"), cât și din punct de vedere al programului (memorie program minimă), cu respectarea restricțiilor de timp;

b) la nivel de coordonator, într-un sistem multi-microcontroler, optimizarea alocării resurselor și realizarea unei comunicații eficiente între componentele sistemului.

La ambele niveluri proiectantul trebuie să poată realiza expandarea, după necesități, a resurselor de memorie sau porturi I/E, eventual cuplarea cu dispozitive de la microprocesoarele de uz general: porturi, controlere DMA, interfețe cu operatorul etc.

Deși în prezent există o mare varietate de microcontrolere, în clasa de 8 biți, produsele firmelor Intel și Motorola au căpătat cea mai mare răspândire. În principal, la dezvoltarea produselor lor, aceste firme au plecat de la experiența obținută în realizarea microprocesoarelor de uz general. Mai mult, s-a căutat să se mențină cât mai mult conceptele introduse odată cu diferitele tipuri de microprocesoare. De exemplu, firma *Intel* a dezvoltat prima sa familie de microcontrolere (MCS-48) având la bază principiile sistemelor cu microprocesoare 8080 și 8085. Deși familia MCS-48 nu mai este un produs de mare actualitate, prin conceptele dezvoltate, aceasta a fost de la început orientată pentru aplicații "împachetate" ("embedded applications") de control [49]. Apoi, Intel a părăsit linia menționată mai sus și, prin familia MCS-51, a elaborat o arhitectură puternică, cunoscută prin produsul reprezentativ 8051, care este și în prezent foarte apreciată. Din acest motiv, diferiți producători au preluat nucleul acestei familii pe care l-au dezvoltat, adăugându-i noi facilități și obținând astfel produse deosebit de performante. Spre exemplu, firma *Siemens* a introdus familia SAB 80C515/535, cu un număr sporit de porturi paralele față de familia de bază MCS-51 (6 porturi I/E, față de 4 la 8051), un convertor A/D de 8 biți cu posibilitatea de creștere prin soft la 10 biți, ceas de gardă (WDT) și un al 3-lea timer cu facilități de captură/comparare. *Philips*, prin familia 80C51, a dezvoltat un produs mult utilizat în prezent (80C552), prin introducerea facilității de comunicație în rețea I²C, a unui convertor A/D pe 12 biți și a unor ieșiri de tip PWM. O dezvoltare interesantă, foarte atractivă pentru utilizatori, este cea adusă de firma *Atmel*, care, prin folosirea de memorie FLASH pe cip a crescut puternic performanțele familiei MCS-51, simplificând totodată proiectarea/dezvoltarea sistemelor. Drept urmare și Intel și-a perfecționat produsele cuprinse în familia MCS-51, prin introducerea de noi facilități "on chip" – 8052, 80C51, 8xC51FA/FB/FC etc., sau chiar prin trecerea (în 1997) la o arhitectură mai avansată - MCS-151.

Toate aceste produse au la bază aceleași resurse software, definite de Intel la MCS-51, ceea ce constituie un alt mare avantaj și fac ca, deși introdusă încă din 1980, arhitectura MCS-51 să se bucure de o mare popularitate și în prezent.



Familia de microcontrolere MCS-51 (8051)

Familia de microcontrolere MCS-51 este în prezent una dintre cele mai cunoscute și utilizate, datorită îmbunătățirii arhitecturii și creșterii performanțelor în raport cu familia MCS-48 [49], [54]. Prin MCS-51 Intel a stabilit un standard în categoria microcontrolerelor de 8 biți, dezvoltând în decurs de două decenii noi generații, toate având ca nucleu aceeași arhitectură. În tab.6.1 sunt prezentate principalele componente ale familiei, împreună cu resursele "on chip" și tehnologia utilizată [50], [51], [54].

Microcontroler	Versiunea fără ROM	Versiunea cu EPROM	ROM	RAM	Timere (16 biți)	Înteruperi	Tehnologie
8051AH	8031AH	8751H, 8751BH	4K × 8	128 × 8	2	5	HMOS
8052AH	8032AH	8752BH	8K × 8	256 × 8	3	6	HMOS
80C51BH	80C31BH	87C51	4K × 8	128 × 8	2	5	CHMOS

Tab.6.1. Principalele componente ale familiei MCS-51

Microcontrolerele 8052 sunt o versiune îmbunătățită a seriei 8051, cu care este complet compatibilă "în jos".

6.1. Caracteristicile generale ale familiei MCS-51

Principalele caracteristici ale arhitecturii de bază 8051 (8052) sunt [50]:

- unitate centrală de procesare de 8 biți, optimizată pentru aplicații de control;
- procesor boolean inclus, cu capabilități extinse pentru prelucrări logice la nivel de bit;
- spațiu de adresare pentru memoria program de 64 Ko;
- spațiu de adresare pentru memoria de date de 64 Ko;
- 4 bancuri de registre de lucru ("scratchpad registers");
- 128 de fanioane configurabile prin program, de către utilizator (128 User-defined SW flags), reprezentând și memoria RAM a procesorului boolean;
- 21 (26) registre cu funcții speciale (SFR - Special Function Registers) pentru controlul resurselor "on chip", dintre care 11 (12) adresabile la nivel de bit;
- 32 linii bidirecționale, sub forma a 4 porturi I/E de 8 biți, adresabile și la nivel de bit (la versiunea fără ROM intern numai 16 linii);
- 2 (3) numărătoare de evenimente/timere de 16 biți;
- un port serial asincron (UART) pentru comunicații "full duplex", inclusiv de tip multiprocesor;
- 5 (6) întreruperi cu două niveluri de prioritate;
- frecvență de lucru între 3,5 și 12MHz, cu versiuni de 16 și 20MHz;
- set de 111 instrucțiuni (dintre care 64 de un singur ciclu mașină), incluzând și instrucțiuni de înmulțire și împărțire;
- un ciclu instrucțiune tipic de 1μs, înmulțiri și împărțiri de întregi în 4μs (la 12MHz);
- aritmetică binară și zecimală;

- stivă internă de 128 de octeți.

Ulterior, firma Intel a extins seria de microcontrolere în tehnologie CHMOS [54], care a fost optimizată pentru aplicații de control ce necesită consum redus de energie, un grad mare de integrare și performanțe ridicate. Astfel, cea mai nouă variantă din familia "clasică" 8051, seria 8xC51FA/FB/FC ($x = 0, 3$ sau 7) conține pe cip 8/16/32 Ko ROM/EPROM, 256 octeți RAM, 3 timere de 16 biți, un grup de numărătoare programabile (Programmable Counter Array) cu: ieșiri de mare viteză, comparare/captură, ieșiri de tip impulsuri modulate în durată (PWM), ceas de gardă (WDT) ș.a. Toate aceste îmbunătățiri au fost preluate și de arhitecturile avansate, produse mai recent: MCS-151 și MCS-251 [53]. Variante CHMOS există și pentru seria 8x52: 8xC52/54/58, cu 8/16/32 Ko ROM/EPROM.

Seria 8xC51RA/RB/RC ($x = 0, 3$ sau 7) [54] beneficiază de un spațiu extins de memorie RAM internă, de 512 octeți. De asemenea, există variante avansate ale seriilor clasice, destinate aplicațiilor cu cerințe speciale. Spre exemplu, seria 8xC51GB ($x = 0, 3$ sau 7) dispune de 8Ko ROM/OTP ROM, 256 octeți RAM, două zone de numărătoare programabile, 3 timere de 16 biți, 15 surse de întrerupere (7 externe și 8 interne), programabile pe patru nivele de prioritate, 48 de linii I/E programabile (6 porturi de 8 biți) cu 40 de intrări de tip trigger Schmitt, convertor A/D de 8 biți cu 8 canale [54].

Variantele de consum redus: 8xL51FA/FB/FC, $x = 0, 3, 7$, 8xL52/54/58, $x = 0, 7$, operând cu V_{CC} între 2,7V și 3,6V, la o frecvență maximă de 20MHz, permit proiectanților să reducă substanțial consumul de energie. O altă direcție de dezvoltare o reprezintă seria MCS-51 de mare viteză care, în loc de frecvențe maxime de 12 sau 16MHz, au limita de 24 sau 33MHz, ceea ce asigură practic dublarea performanțelor [54].

Toate realizările menționate mai sus au aceeași arhitectură și același set de instrucțiuni caracteristic familiei MCS-51, iar majoritatea sunt și pin cu pin compatibile. Această compatibilitate "pe verticală" oferă un mare avantaj proiectanților de sisteme bazate pe astfel de microcontrolere, întrucât cunoașterea caracteristicilor de bază asigură o modernizare rapidă și ușoară a produselor realizate cu MCS-51.

6.2. Conexiunile externe ale familiei MCS-51

Utilizarea pe scară largă a pinilor cu mai multe funcții stă la baza compatibilității pin cu pin a microcontrolerelor din familia MCS-51. Excepție fac microcontrolerele destinate aplicațiilor speciale, care necesită de regulă capsule cu mai mulți pini, având funcții dedicate. În fig.6.1 se prezintă simbolul logic al microcontrolerelor din familia MCS-51 (fig.6.1a), precum și semnalele la pini pentru principalele trei tipuri de capsule folosite în aplicații (fig.6.1b, 6.1c și 6.1d). Pinii marcați prin "N.U." nu sunt utilizați și trebuie lăsați neconectați.

Semnificația semnalelor la pini este următoarea:

XTAL1 - intrarea amplificatorului oscilatorului intern și intrarea de sincronizare a blocului intern de secvențiere a operațiilor, este folosită pentru conectarea rezonatorului extern cu cuarț sau ceramic.

XTAL2 - ieșirea amplificatorului oscilatorului intern. La acest pin se poate conecta un semnal de tact extern sau un rezonator cu cuarț ori ceramic.

RST (ReSeT) - linie de intrare, prin activarea căreia, pe durata a două cicluri mașină (în prezența semnalului de tact), se realizează resetarea circuitului. Aceasta se poate obține folosind doar un condensator extern către masă (V_{SS}), întrucât intrarea este deja prevăzută cu un rezistor intern către V_{DD} ("pullup" resistor).

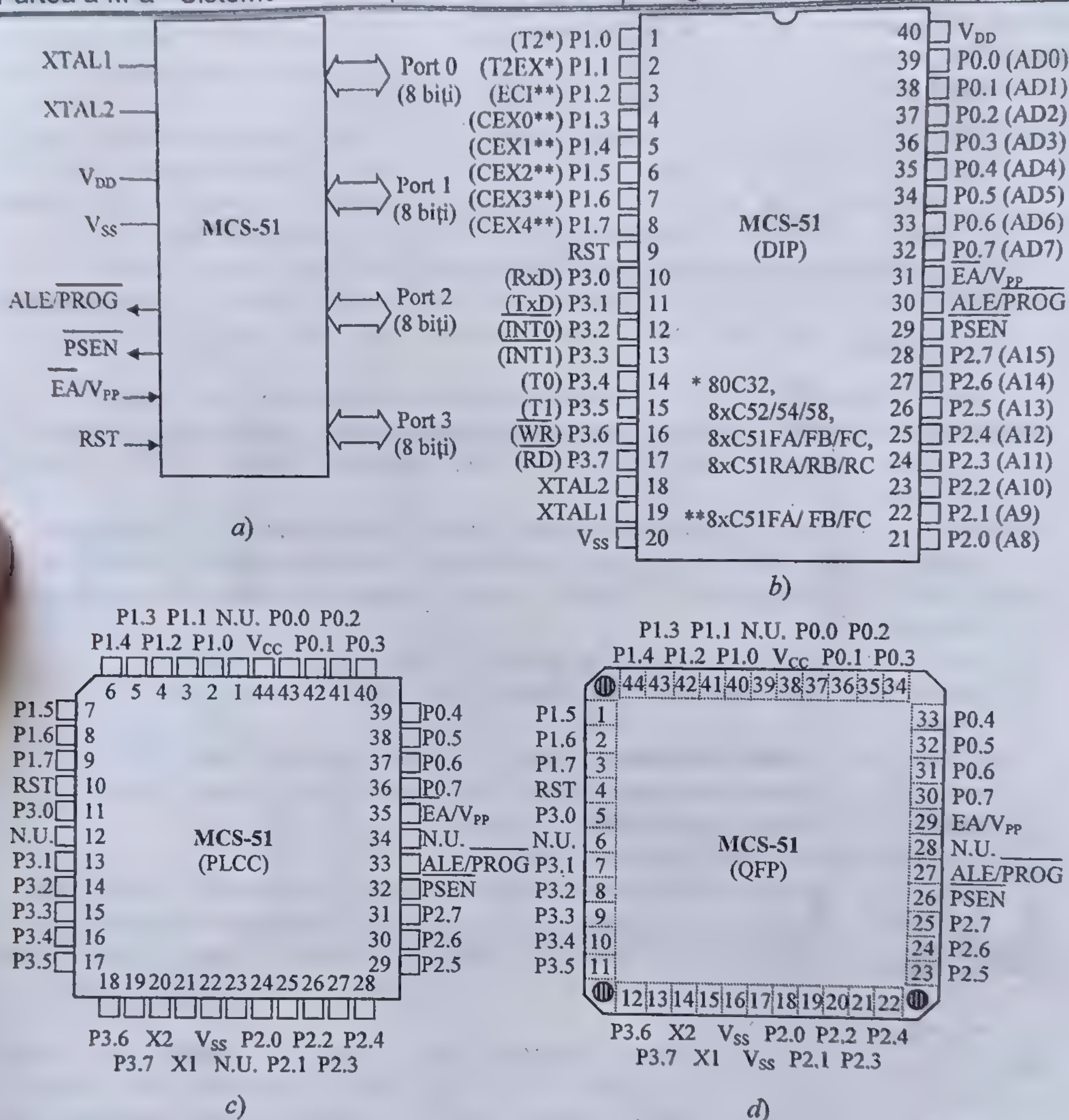


Fig.6.1. Familia MCS-51 - simbolul logic (a) și semnalele la pini (b, c, d)

\overline{EA}/V_{PP} (External Access / V_{PP}) - la microcontrolerele care au memorie program internă (de ex. 8051), acest pin are rol de intrare de validare (pe "0" logic) a accesului numai la memoria program externă (0000÷FFFFh). Atunci când este pus la "1" logic, 8051 extrage și execută instrucțiuni din memoria ROM internă dacă adresa din PC este mai mică de 1000h. Pentru microcontrolerele care nu au memorie program internă (spre ex. 8031), acest pin trebuie conectat la masă (V_{SS}). La microcontrolerele care au memorie program internă de tip EPROM în timpul programării acesteia pe acest pin se aplică tensiunea de programare corespunzătoare (vezi §6.3.8).

P0.0÷P0.7 (Port 0) - port I/E de 8 biți, bidirecțional, cu drenă în gol. Poate fi folosit ca port de I/E de uz general sau pentru accesul la memoria externă: furnizează, multiplexat în timp, octetul inferior al adresei (A0÷A7) și octetul de date (D0÷D7).

P1.0÷P1.7 (Port 1) - port I/E de 8 biți, bidirecțional, de uz general. La microcontrolerele 80C32, 8xC52/54/58, 8xC51FA/FB/FC și 8xC51RA/RB/RC pot fi utilizate funcțiile alternative ale acestor pini (vezi și fig.6.1b), caz în care biții registrului de ieșire asociat trebuie poziționați pe "1" logic. Funcțiile secundare ale Portului 1 sunt următoarele:

T2 (P1.0) - intrare de numărare pentru Timerul 2 sau ieșire semnal de tact (Clock Out);

T2EX (P1.1) - intrare de control (captură, reîncărcare, sens de numărare) pentru Timerul 2;

ECI (P1.2) - intrare de numărare pentru timerul zonei de numărătoare programabile (PCA);

CEX0÷CEX4 (P1.3÷P1.7) - I/E pentru modulele de comparare/captură 0÷4 ale PCA.

P2.0÷P2.7 (Port 2) - port I/E de 8 biți, bidirecțional. Poate fi folosit ca port de I/E de uz general, sau pentru adresarea memoriei externe, când furnizează octetul superior al adresei (A8÷A15).

P3.0÷P3.7 (Port 3) - port de 8 biți, bidirecțional. Poate fi utilizat ca port de I/E de uz general, sau pot fi utilizate funcțiile secundare ale acestor pini, caz în care biții registrului de ieșire asociat trebuie poziționați pe "1" logic. Funcțiile secundare ale portului 3 sunt următoarele (vezi și fig.6.1b):

RxD (P3.0) - intrare de date serială asincronă sau I/E de date serială sincronă;

TxD (P3.1) - ieșire de date serială asincronă sau ieșire semnal de tact pentru modul sincron;

INT0 (P3.2) - intrare de întrerupere externă 0 sau intrare de control pentru Timerul 0;

INT1 (P3.3) - intrare de întrerupere externă 1 sau intrare de control pentru Timerul 1;

T0 (P3.4) - intrare de numărare pentru Timerul 0;

T1 (P3.5) - intrare de numărare pentru Timerul 1;

WR (P3.6) - ieșire de comandă pentru înscrierea unui octet în memoria de date externă;

RD (P3.7) - ieșire de comandă pentru citirea unui octet din memoria de date externă.

PSEN (Program Store Enable) - ieșire de comandă a citirii memoriei program externe. Este activată o dată la fiecare 6 perioade ale oscilatorului, exceptând ciclurile de acces la memoria externă de date.

ALE/PROG (Address Latch Enable / PROGramming) - ieșire de comandă a memorării octetului inferior al adresei, depus pe liniile AD0÷AD7 în timpul ciclurilor de acces la memoria externă (funcția ALE). Opțiunea **PROG** se utilizează ca intrare pentru impulsul de programare, la programarea EPROM-ului intern, acolo unde este cazul (vezi §6.3.8).

V_{DD} - alimentare cu tensiune (+5Vc.c.).

V_{SS} - conectare la masă (0V).

6.3. Arhitectura familiei MCS-51

În fig.6.2 este prezentată schema bloc internă a familiei de microcontrolere MCS-51 pentru cele două tipuri de bază: 8051 și 8052. Se folosește o arhitectură internă Harvard [50], [51], cu magistrale distincte pentru adresarea memoriei RAM interne (RAR - RAM Address Register) și a memoriei program interne (PAR - Program Address Register) de tip ROM sau EPROM.

Microcontrolerele au o unitate centrală de prelucrare (CPU) bazată pe acumulator (ACC), cu o unitate logico-aritmetică (ALU) de 8 biți și un număr limitat de registre la

dispoziția utilizatorului: B, PSW (Program Status Word) și SP (Stack Pointer), toate de 8 biți, precum și două registre de 16 biți: PC (Program Counter) și DPTR (Data Pointer).

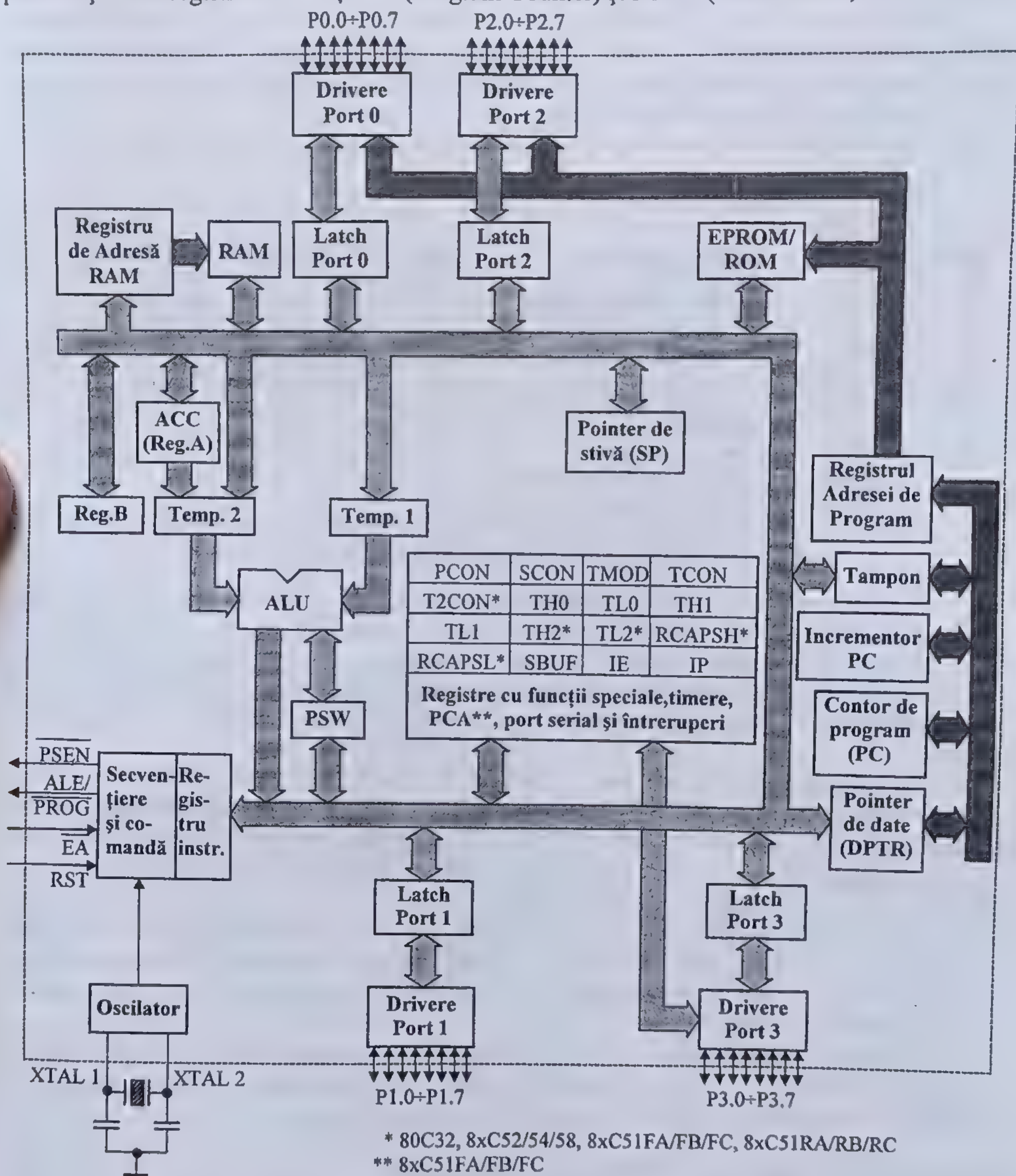


Fig.6.2. Schema bloc internă a microcontrolerelor din familia MCS-51

Registrul B este folosit pe durata operațiilor de înmulțire și împărțire sau ca registru de uz general pentru celelalte tipuri de instrucțiuni. Registrul pointer de date, DPTR, este singurul registru de 16 biți destinat adresării indirecte a variabilelor din memoria externă de date. Acest registru are două componente: DPH - pentru octetul superior și DPL - pentru octetul inferior,

cea ce îi conferă facilitatea de a fi utilizat fie ca registru de 16 biți, fie ca două registre independente de 8 biți.

Registrul cuvântului de stare a programului, PSW, poate fi adresat și la nivel de bit și conține informații referitoare la indicatorii de condiții și biți de selecție a bancurilor de registre de lucru, amplasate în memoria RAM internă (v. §6.2.2.2). În fig.6.3 este detaliată configurația acestui registru, care are adresa absolută D0h.

(msb)						(lsb)		
CY	AC	F0	RS1	RS0	OV	-	P	(D0h)

Simbol	Poziția	Nume și semnificație
CY	PSW.7	Fanion de transport
AC	PSW.6	Fanion de transport auxiliar (pentru operații în BCD)
F0	PSW.5	Fanion 0 (de uz general)
RS1	PSW.4	Biți pentru selecția bancului cu registre de lucru (v. Nota)
RS0	PSW.3	
OV	PSW.2	Fanion de depășire
-	PSW.1	Fanion definibil de utilizator
P	PSW.0	Fanion de paritate

Notă: Valorile biților (RS1, RS0) selectează în următorul mod bancurile de registre:

(0,0) - Banc 0	(00h - 07h)
(0,1) - Banc 1	(08h - 0Fh)
(1,0) - Banc 2	(10h - 17h)
(1,1) - Banc 3	(18h - 1Fh)

Fig.6.3. PSW - registrul cuvântului de stare a programului

O caracteristică importantă a unității logico-aritmetice (ALU) la această familie este faptul că poate manipula nu numai date de un octet ci și de un bit, datorită unei logici incluse denumită *procesor boolean*. Această facilitate, neîntâlnită la alte familii de microcontrolere, permite implementarea extrem de avantajoasă a structurilor de tip automat programabil la nivel de bit. Setul de instrucțiuni este prevăzut cu un subset important pentru manipularea variabilelor booleene.

Arhitectura de tip Harvard este utilizată și în cazul în care CPU lucrează cu memorie externă. În acest caz, prin intermediul registrelor PC și DPTR pot fi adresate spații distincte de program și de date, fiecare de până la 64 Ko. Ca și în cazul familiei MCS-48, se utilizează semnale cu funcții și semnificații similare: ALE, $\overline{\text{PSEN}}$, $\overline{\text{EA}}$ [49]. Conținutul registrelor PC și DPTR este transmis în exterior prin Porturile 0 și 2.

La inițializarea HW (prin activarea semnalului RST), registrul SP se poziționează implicit la adresa 07h din memoria RAM internă, ca și la microcontrolerele din familia MCS-48. Prin modificarea conținutului registrului SP, stiva poate fi organizată în orice altă zonă a memoriei interne de date.

Celelalte resurse interne: timerele, interfața serială, semnalele $\overline{\text{RD}}$ și $\overline{\text{WR}}$ pentru memoria externă de date, precum și cele două linii de întrerupere externe pot fi evidențiate în exterior prin intermediul Portului 3 (parțial și Portul 1 pentru 8052).

6.3.1. Organizarea și configurarea memoriei

Arhitectura familiei MCS-51 suportă mai multe spații de adresare, fizic distincte, separate funcțional la nivel hardware prin mecanisme diferite de adresare și prin semnale distincte de citire/scriere:

- memoria program integrată pe cip, de 4Ko sau 8Ko;
- memoria de date internă, de 128 sau 256 octeți;
- memoria program externă, până la 64 Ko;

- memoria de date externă, până la 64 Ko;
- zona de registre cu funcții speciale (Special Function Registers), inclusă pe cip, de 128 de octeți.

În funcție de tipul microcontrolerului (vezi tab.6.1), memoria program internă poate lipsi. Figura 6.4 ilustrează diferitele spații de memorie, semnalele care asigură separarea acestor zone: \overline{EA} și \overline{PSEN} - pentru memoria program, respectiv $\overline{RD}/\overline{WR}$ - pentru memoria de date, împreună cu modurile corespunzătoare de adresare.

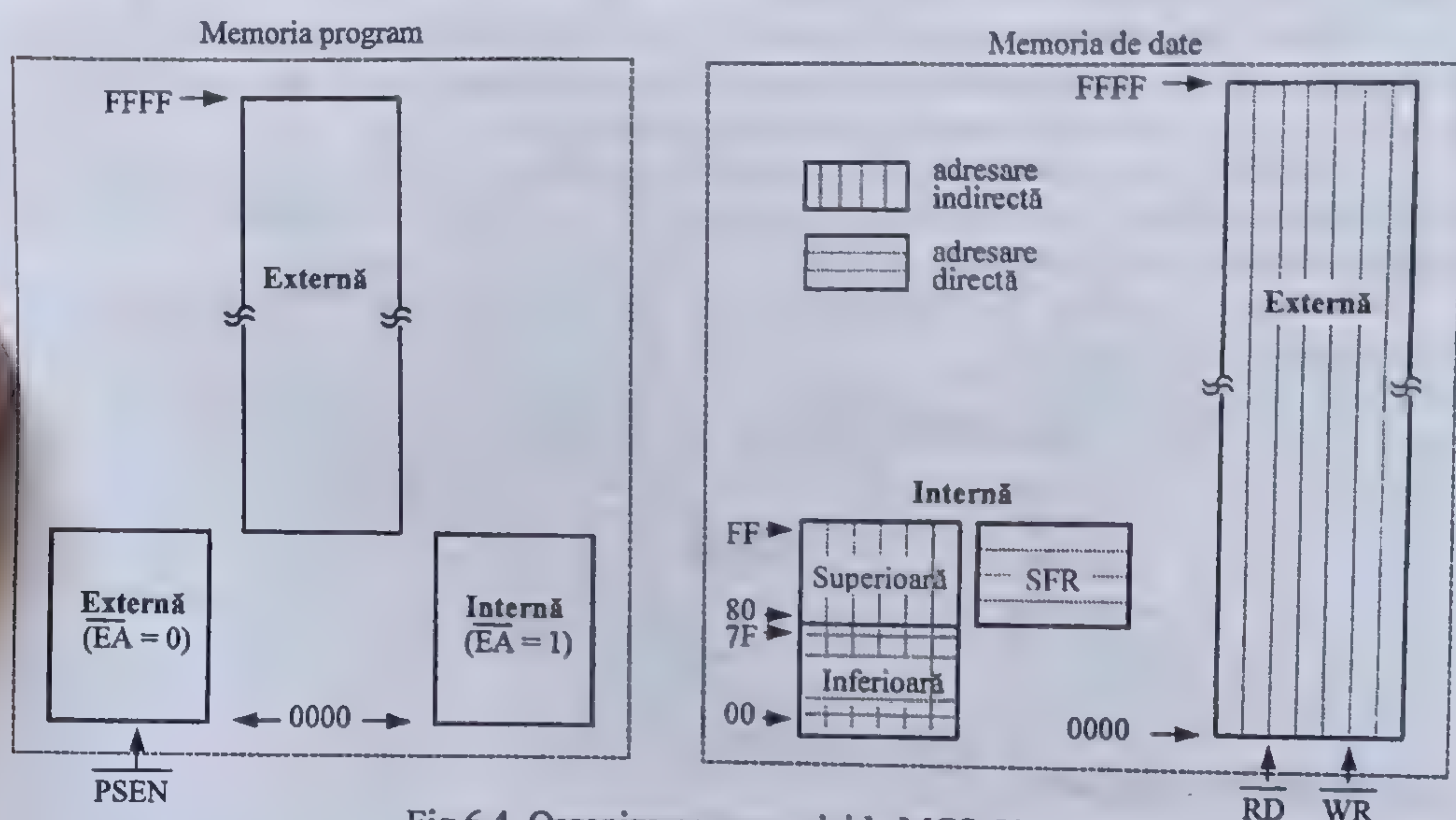


Fig.6.4. Organizarea memoriei la MCS-51

6.3.1.1. Memoria program

După resetarea microcontrolerului, CPU începe execuția programului de la locația 0000h. În zona inferioară a memoriei program sunt rezervate spații de 8 octeți pentru rutinele de servire a întreruperilor externe sau interne, așa cum se prezintă în fig.6.5. Întreruperilor externe 0 și 1 li s-au rezervat locațiile 0003h și 0013h, pentru Timerele 0 și 1 locațiile 000Bh și respectiv 001Bh etc. (v. §6.3.5). Pentru rutinele scurte, specifice aplicațiilor de control, intervalul de 8 octeți este suficient, în caz contrar trebuie prevăzute instrucțiuni de salt la adresele de început ale rutinelor de servire. Dacă una din sursele de întrerupere nu este utilizată, spațiul corespunzător poate fi folosit pentru programul principal.

Primii 4/8/16/32Ko ai memoriei program se pot afla, în funcție de tipul microcontrolerului, fie pe cip, fie în exterior. Selecția se realizează cu linia \overline{EA} , prin conectare la V_{CC} sau la V_{SS} .

La dispozitivele cu 4Ko ROM/EPROM pe cip, dacă \overline{EA} se conectează la V_{CC} atunci programul de la adresele 0000h până la 0FFFh se va executa direct din memoria internă,

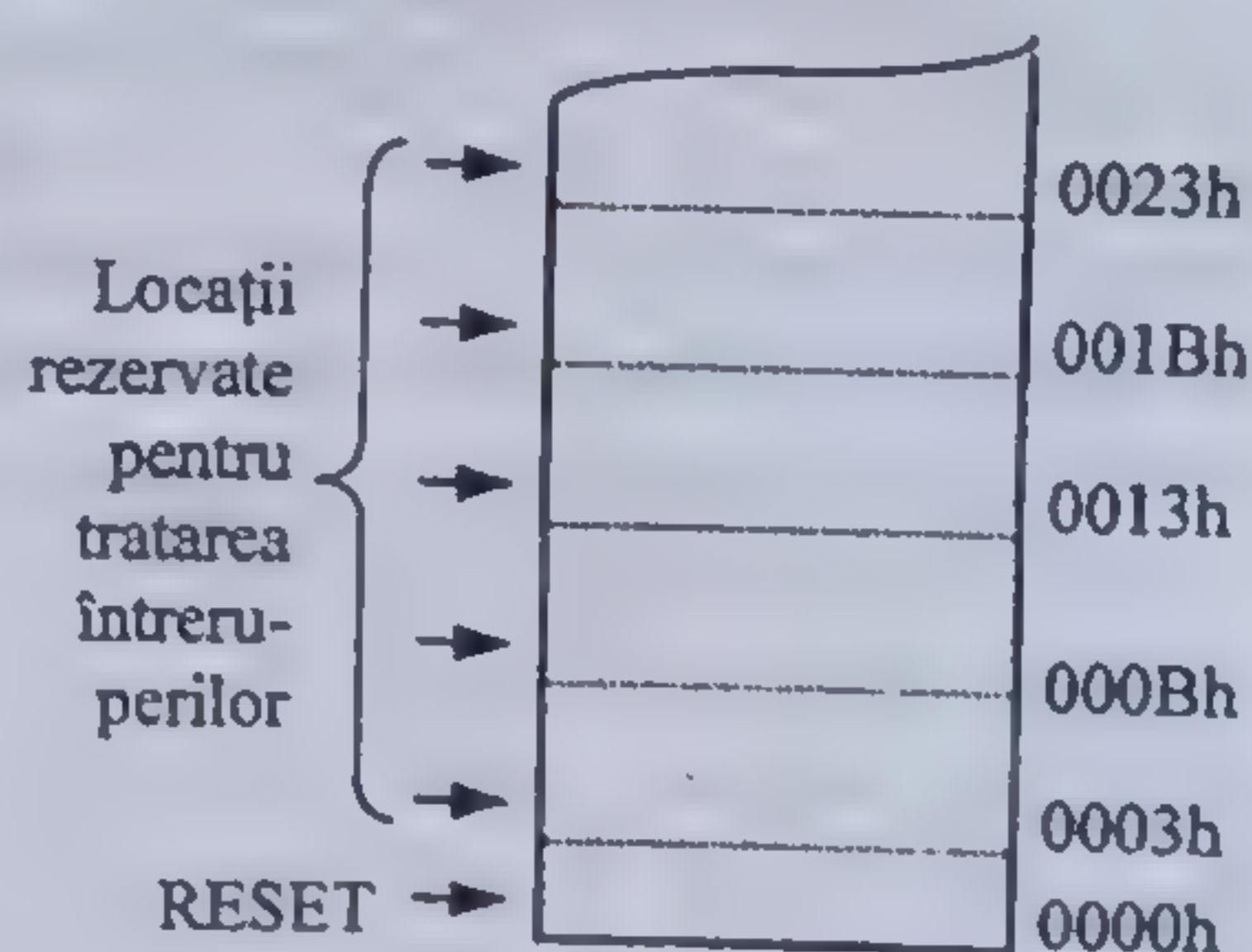


Fig.6.5. Harta zonei inferioare a memoriei program

memoria externă (dacă există) fiind neutilizată. De la adresa 1000h până la adresa FFFFh se va executa programul din memoria externă. La microcontrolerele cu 8/16/32 Ko, $\overline{EA} = V_{CC}$ validează accesul la memoria program internă în spațiul de adrese cuprins între 0000h și 1FFFh/3FFFh/7FFFh, iar spațiul de la 2000h/4000h/8000h până la FFFFh corespunde memoriei program externe.

Dacă $\overline{EA} = V_{SS}$, atunci întregul program se va executa din memoria externă, memoria program internă (dacă există) fiind dezactivată.

Citirea din memoria program externă se face prin activarea semnalului \overline{PSEN} , la extragerea codurilor instrucțiunilor sau la execuția unor instrucțiuni MOV_C (MOVE Code byte). În cazul citirii din memoria program internă, acest semnal nu este activat.

În fig.6.6 este prezentat modul de configurare hardware a memoriei program externe, asemănător cu cel de la familia MCS-48. Porturile 0 și 2, așa cum s-a menționat, sunt dedicate magistralei externe de adrese/date.

Portul 0 are rolul de magistrală externă de date pentru aducerea din exterior a instrucțiunilor. În prima fază a ciclului, pe liniile Portului 0 este depus octetul inferior al adresei, care se reține într-un latch (8212) cu ajutorul semnalului ALE. Apoi P0 este eliberat și, odată cu activarea liniei \overline{PSEN} , se citește memoria.

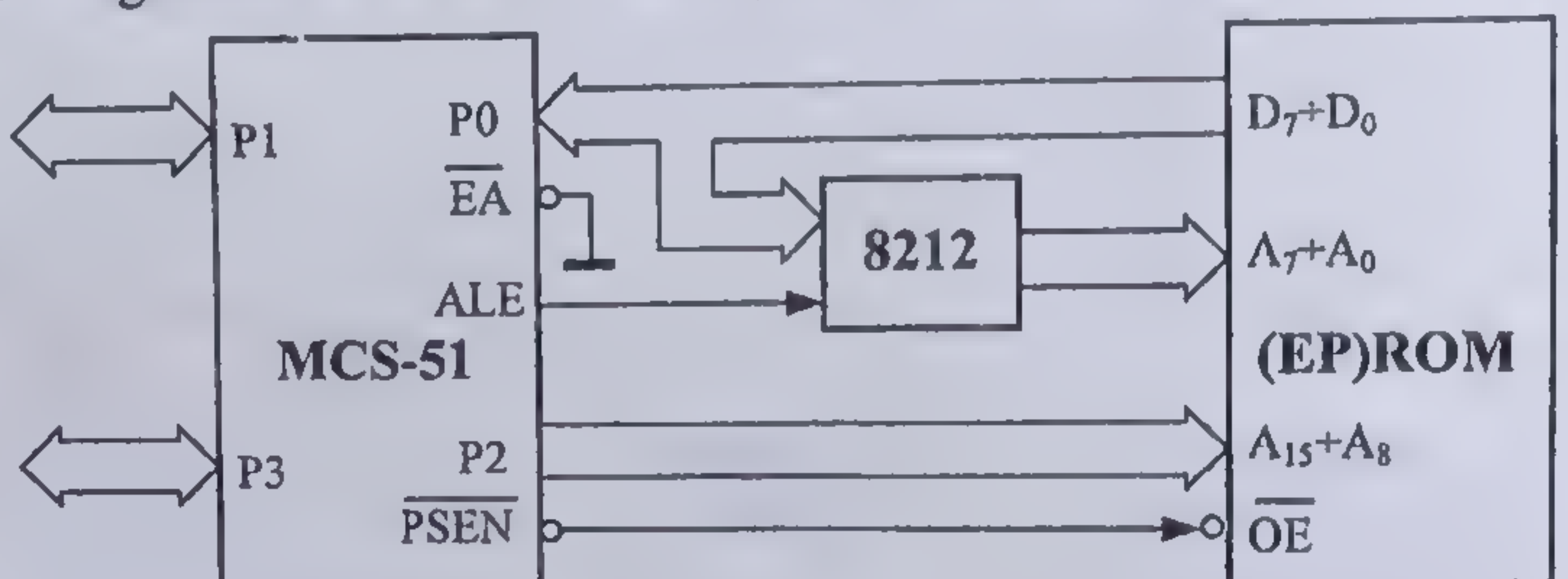


Fig.6.6. Schema de configurare a memoriei program externe

6.3.1.2. Memoria de date

În fig.6.4 este prezentat și spațiul memoriei de date accesibil utilizatorului, în interiorul și exteriorul microcontrolerelor familiei MCS-51.

Memoria de date internă se compune dintr-un spațiu divizat în următoarele trei blocuri: a) zona inferioară de 128 octeți, b) zona superioară de 128 octeți (numai la 8052) și c) zona SFR (Special Function Registers), tot de 128 octeți. Deși zona superioară de RAM intern și zona SFR ocupă același spațiu de adrese, între 80h și FFh, acestea sunt entități fizice separate, fiind accesibile prin moduri de adresare diferite (v.fig.6.4).

Cei 128 de octeți ai zonei inferioare, cu adrese de octet cuprinse între 00h și 7Fh, pot fi adresați atât direct, cât și indirect. Primii 32 de octeți grupează 4 bancuri a câte 8 registre generale de lucru de 8 biți. Bancul de registre curent, R0÷R7, poate fi selectat dintre acestea prin biții RS1, RS0 ai registrului PSW (fig.6.3). În fig.6.7a este detaliată harta acestei zone a memoriei RAM interne. După resetarea microcontrolerului este selectat implicit bancul 0 (00h÷07h), iar registrul SP se poziționează la adresa 07h. Următorii 16 octeți, dispuși deasupra celor patru bancuri, formează un spațiu RAM adresabil atât la nivel de octet, cât și la nivel de bit, fiind utilizat de procesorul boolean. Cei 128 de biți ai acestui spațiu pot fi folosiți și ca fanioane de către utilizator, având adresele de bit cuprinse între 00h și 7Fh (128 User - defined SW flags).

Zona superioară de 128 octeți (v.fig.6.4), cu adrese cuprinse între 80h și FFh, poate fi accesată numai prin adresare indirectă și există numai la microcontrolerele cu RAM intern de 256 octeți (v.tab.6.1). Această parte a memoriei interne nu este adresabilă la nivel de bit și poate

fi folosită fie pentru memorarea datelor, fie ca stivă (adresarea se face indirect, prin intermediul registrului SP).

Zona SFR este un spațiu de RAM cu adresare directă (vezi fig.6.4), destinată registrelor cu funcții speciale din CPU (cu excepția PC) sau a celorlalte resurse de pe cip, în afară de bancurile cu registre de lucru. În fig.6.7b este prezentată harta acestei zone pentru 8051 și 8052, împreună cu adresele specifice. Se observă că această zonă este ocupată numai parțial.

În general, toate microcontrolerele familiei MCS-51 au aceleași registre cu funcții speciale ca 8051 și cu aceleași adrese în spațiul SFR. Variantele îmbunătățite au registre speciale adiționale care nu se regăsesc la seria 8051; la 8052 acestea sunt evidențiate în fig.6.7b.

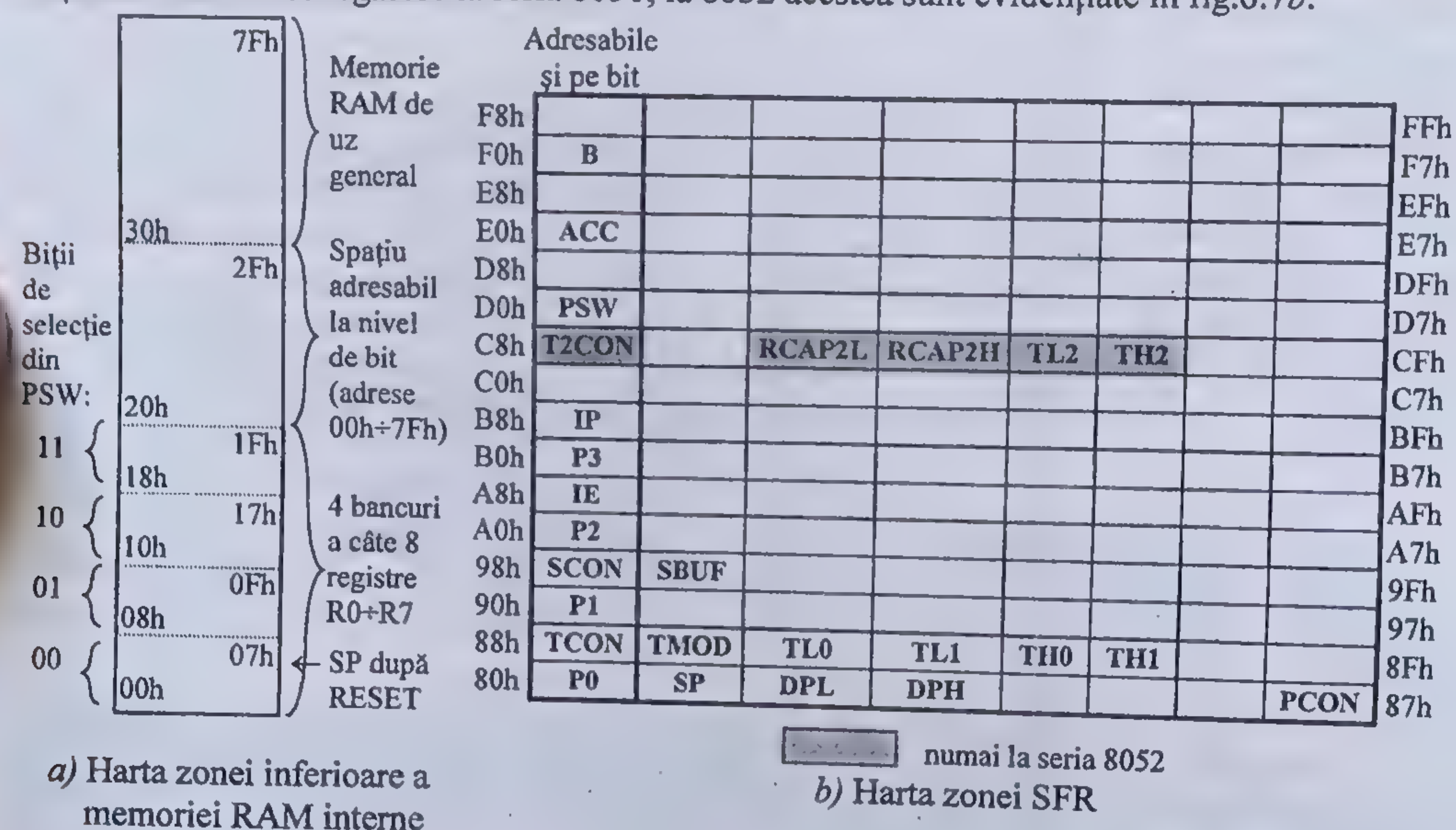


Fig.6.7. Harta memoriei interne de date la MCS-51

O astfel de organizare a SFR permite o extindere facilă a diverselor variante îmbunătățite de microcontrolere. Locațiile neutilizate, dintre cele 128, nu sunt implementate pe cip; citirea lor returnează valori aleatoare, iar scrierea nu are nici un efect. Un număr de 16 locații din spațiul SFR, a căror adresă este multiplu de 8, sunt adresabile atât la nivel de octet cât și la nivel de bit.

În tabelul 6.5 este prezentată semnificația registrelor SFR utilizate la 8051 și 8052. Registrele marcate cu asterisc sunt cele adresabile și la nivel de bit și au adresa de bit specificată în tabel. Registrele A, B, PSW, SP și DPTR ale unității centrale de procesare pot fi utilizate ca atare, cu numele lor în setul de instrucțiuni. Ca registre cu funcții speciale, aceleași locații se pot adresa și direct, folosindu-se în acest scop fie numele de SFR (ACC, B, PSW, SP, respectiv DPL și DPH pentru DPTR), fie adresa lor SFR.

Registrele P0, P1, P2 și P3 sunt SFR-urile corespunzătoare latch-urilor Porturilor 0, 1, 2 și respectiv 3. Similar, registrele pereche (TH0, TL0), (TH1, TL1) și (TH2, TL2) sunt SFR-urile numărătoarelor de 16 biți ale Timerelor 0, 1 și respectiv 2. De asemenea, registrul pereche (RCAP2H, RCAP2L) este SFR-ul Timerului 2 pentru funcționarea în modul special “captură - reîncărcare”. Inițierea unei transmisii seriale se face scriind direct în SBUF. Pentru controlul regimurilor de funcționare a timerelor, a portului serial și a întreruperilor interne/externe există

registrele speciale TMOD, TCON, T2CON, SCON, PCON, IP și IE. Toate aceste registre cu funcții speciale se pot referi prin numele SFR sau prin adresa SFR. Biții celor 11 (12) registre speciale care pot fi adresate și la nivel de bit pot fi desemnați fie prin numele SFR al bitului, fie prin adresa de bit a acestuia. Spre exemplu:

- Acumulatorul: fie ACC.0, ACC.1, ..., ACC.7, fie E0h, E1h, ..., E7h
- PSW: fie PSW.0, PSW.1, ..., PSW.7, fie D0h, D1h, ..., D7h
- P1: fie P1.0, P1.1, ..., P1.7, fie 90h, 91h, ..., 97h
- TCON: fie TCON.0, TCON.1, ..., TCON.7, fie 88h, 89h, ..., 8Fh.

Aceste facilități asigură o flexibilitate deosebită în manipularea resurselor microcontrolerelor familiei MCS-51, toate componentele SFR fiind tratate ca locații de memorie. Semnificația biților registrelor speciale cu adresare pe bit va fi prezentată ulterior, cu ocazia discutării amănunțite a resurselor de pe cip și a setului de instrucțiuni.

Memoria de date externă poate fi adresată numai indirect (v.fig.6.4), prin intermediul registrelor R0 sau R1 din bancul de registre curent ("banked memory model"), sau prin intermediul registrului DPTR. În ambele cazuri se utilizează instrucțiuni de tipul MOVX (MOVe eXternal RAM). Pentru definirea sensului de transfer, se activează unul din semnalele \overline{RD} sau \overline{WR} . În fig.6.8 este prezentată schema de configurare a unei memorii RAM externe de 2Ko în opt pagini de 256

Numele SFR	Adresa SFR	Registre cu funcții speciale	Adresa de bit
*P0	80h	Port 0	80h ÷ 87h
SP	81h	Stack Pointer	—
DPL	82h	Data Pointer, Low Byte	—
DPH	83h	Data Pointer, High Byte	—
PCON	87h	Power Control Register	—
*TCON	88h	Timer Control Register	88h ÷ 8Fh
TMOD	89h	Timer Mode Register	—
TL0	8Ah	Timer 0, Low Byte	—
TL1	8Bh	Timer 1, Low Byte	—
TH0	8Ch	Timer 0, High Byte	—
TH1	8Dh	Timer 1, High Byte	—
*P1	90h	Port 1	90h ÷ 97h
*SCON	98h	Serial Port Control Register	98h ÷ 9Fh
SBUF	99h	Serial Port Buffer Register	—
*P2	A0h	Port 2	A0h ÷ A7h
*IE	A8h	Interrupt Enable Register	A8h ÷ AFh
*P3	B0h	Port 3	B0h ÷ B7h
*IP	B7h	Interrupt Priority Register	B7h ÷ BFh
*T2CON	C8h	Timer-2 Control Register	C8h ÷ CFh
RCAP2L	CAh	Reload/Capture Register, Low Byte	—
RCAP2H	CBh	Reload/Capture Register, High Byte	—
TL2	CCh	Timer 2, Low Byte	—
TH2	CDh	Timer 2, High Byte	—
*PSW	D0h	Program Status Word Register	D0h ÷ D7h
*ACC	E0h	Accumulator	E0h ÷ E7h
*B	F0h	B Register	F0h ÷ F7h

*Registre speciale adresabile pe octet și pe bit

Tab.6.5. Registrele speciale de funcții (SFR)

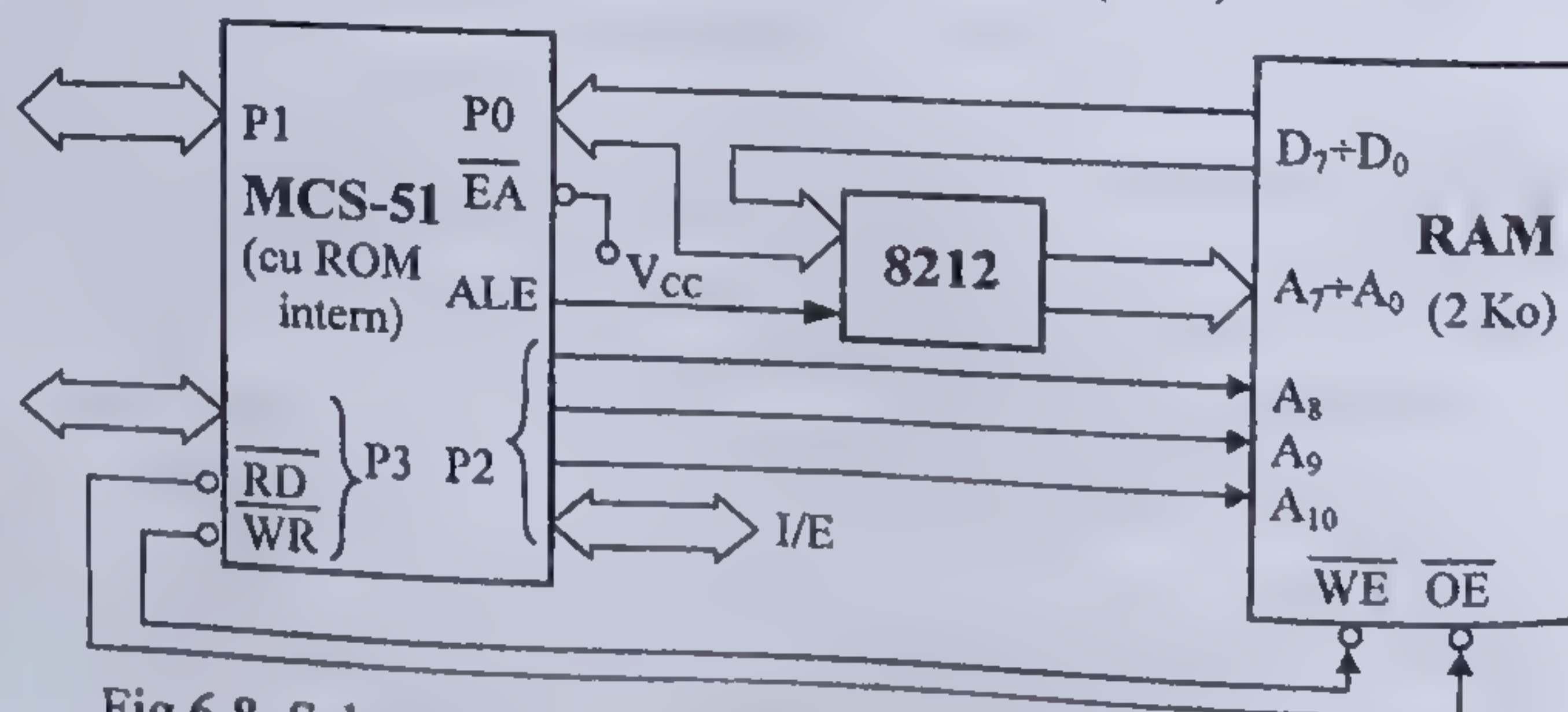


Fig.6.8. Schema de configurare a memoriei RAM externe la MCS-51 cu memorie program internă

octeți fiecare. Pentru definirea paginilor se folosesc trei linii ale Portului 2. Portul 0 transmite multiplexat în timp adresa de 8 biți dintr-o pagină, care este reținută cu ALE într-un latch 8212. Se presupune că CPU execută un program din ROM-ul intern ($\overline{EA}=V_{CC}$). În acest caz, liniile Portului 2 neutilizate la paginare pot fi folosite ca linii I/E. Dacă adresarea se realizează cu registrul DPTR (adrese de 16 biți), atunci toate liniile Portului 2 sunt folosite pentru transmiterea octetului superior al adresei.

Există posibilitatea ca în memoria RAM externă să fie dispus și programul (avantajos pentru faza de dezvoltare a sistemului), caz discutat în paragraful următor.

6.3.2. Secvențierea operațiilor interne la MCS-51

Generatorul intern de tact, pilotat de oscilatorul intern conectat prin XTAL1 și XTAL2 (v.fig.6.2) la un rezonator extern, definește succesiunea operațiilor interne. O stare mașină este formată din două perioade ale oscilatorului intern XTAL2, numite faze: P1, P2 (Phase 1, Phase 2). Un ciclu mașină este compus din 6 stări: S1, S2, ..., S6. Astfel, un ciclu mașină are 12 perioade ale oscilatorului intern, notate S1P1, S1P2, S2P1, ..., S5P2, S6P1, S6P2. Prin urmare la o frecvență a oscilatorului de 12 MHz, durata unui ciclu mașină este de 1 μ s. Din instrucțiunile cu care este dotată familia MCS-51, peste 50% (64 instrucțiuni) sunt de 1 ciclu mașină, celelalte necesitând 2 cicluri mașină. Excepție fac numai cele două instrucțiuni de multiplicare și împărțire, care necesită câte 4 cicluri mașină (4 μ s @ 12 MHz).

6.3.2.1. Secvențierea instrucțiunilor de 1 ciclu mașină

În fig.6.9 sunt prezentate cronogramele pentru extragerea și execuția instrucțiunilor de un ciclu mașină. Pentru execuția unei instrucțiuni într-un ciclu mașină sunt generate două cicluri fetch: primul în S1, iar al doilea în S4. Această procedură e necesară deoarece o parte a instruc-

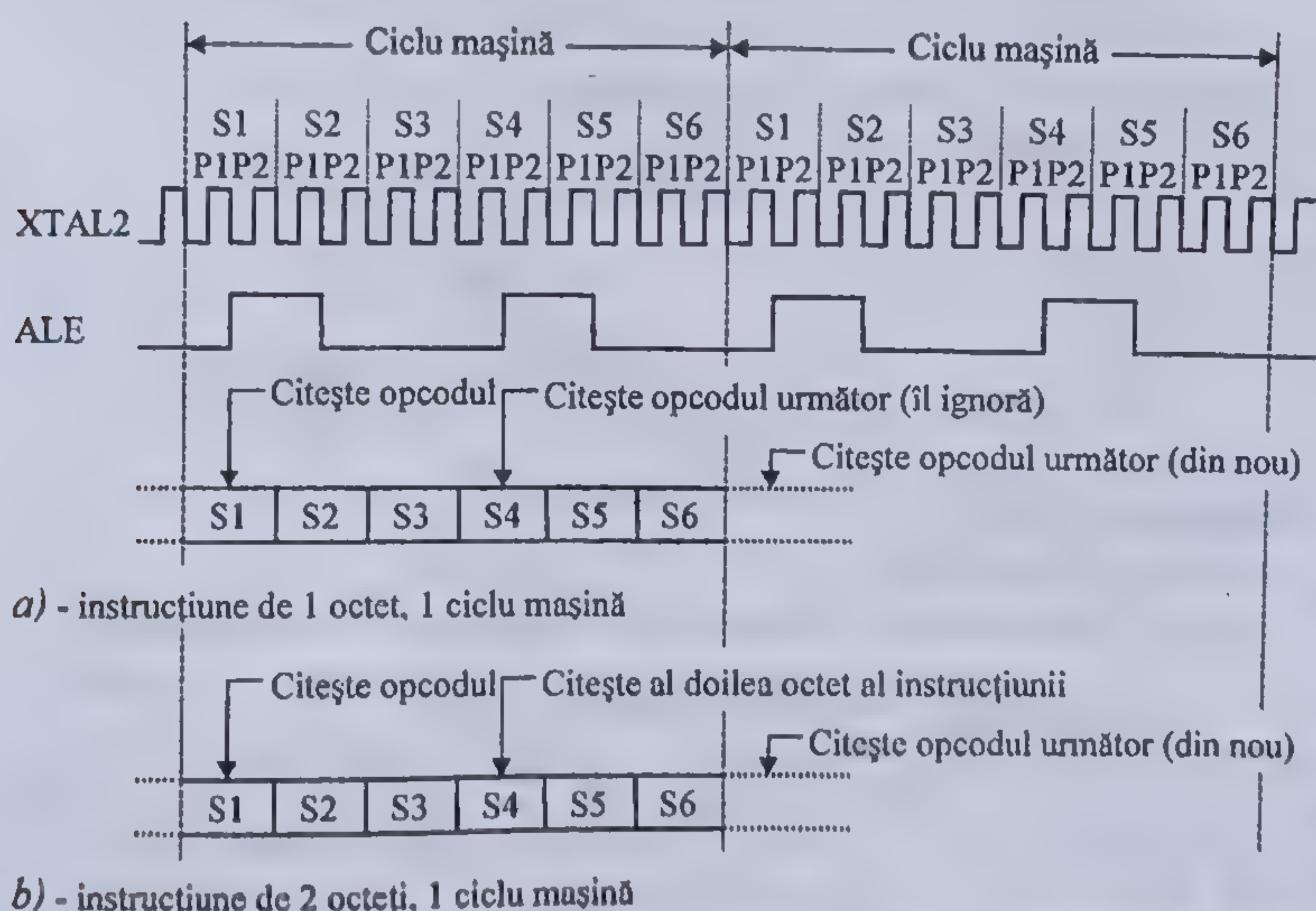


Fig.6.9. Extragerea și execuția instrucțiunilor de 1 ciclu mașină

țiunilor de 1 ciclu mașină sunt de 2 octeți. În cazul instrucțiunilor de un octet și de un ciclu mașină, cea de a doua operație fetch, din S4, nu se realizează (fig.6.9a). În cazul instrucțiunilor

de 2 octeți, codul instrucțiunii extras în S1 este reținut în registrul instrucțiunii (v.fig.6.2) și apoi, în S4, este citit al doilea octet. Până la sfârșitul stării S6 instrucțiunea este executată complet (fig.6.9b).

6.3.2.2. Secvențierea instrucțiunilor de 2 cicluri mașină

Există instrucțiuni de un singur octet care, datorită complexității operațiilor interne, necesită 2 cicluri mașină pentru execuție. La rândul lor acestea pot avea două evoluții, în funcție de tipul operației: dacă necesită sau nu acces la memoria externă de date.

Astfel, pentru instrucțiunile care implică *numai operații interne, fără apel la memoria de date externă* (de exemplu, incrementarea registrului DPTR: INC DPTR) evoluția este cea din fig.6.10a. Se citește opcodul numai în starea S1 a primului ciclu, celelalte citiri succesive fiind ignorate.

Al doilea tip de evoluție este cel specific instrucțiunilor de acces la memoria RAM externă, de tip MOVX (fig. 6.10b). Și în acest caz opcodul se citește în starea S1 a ciclului 1, dar

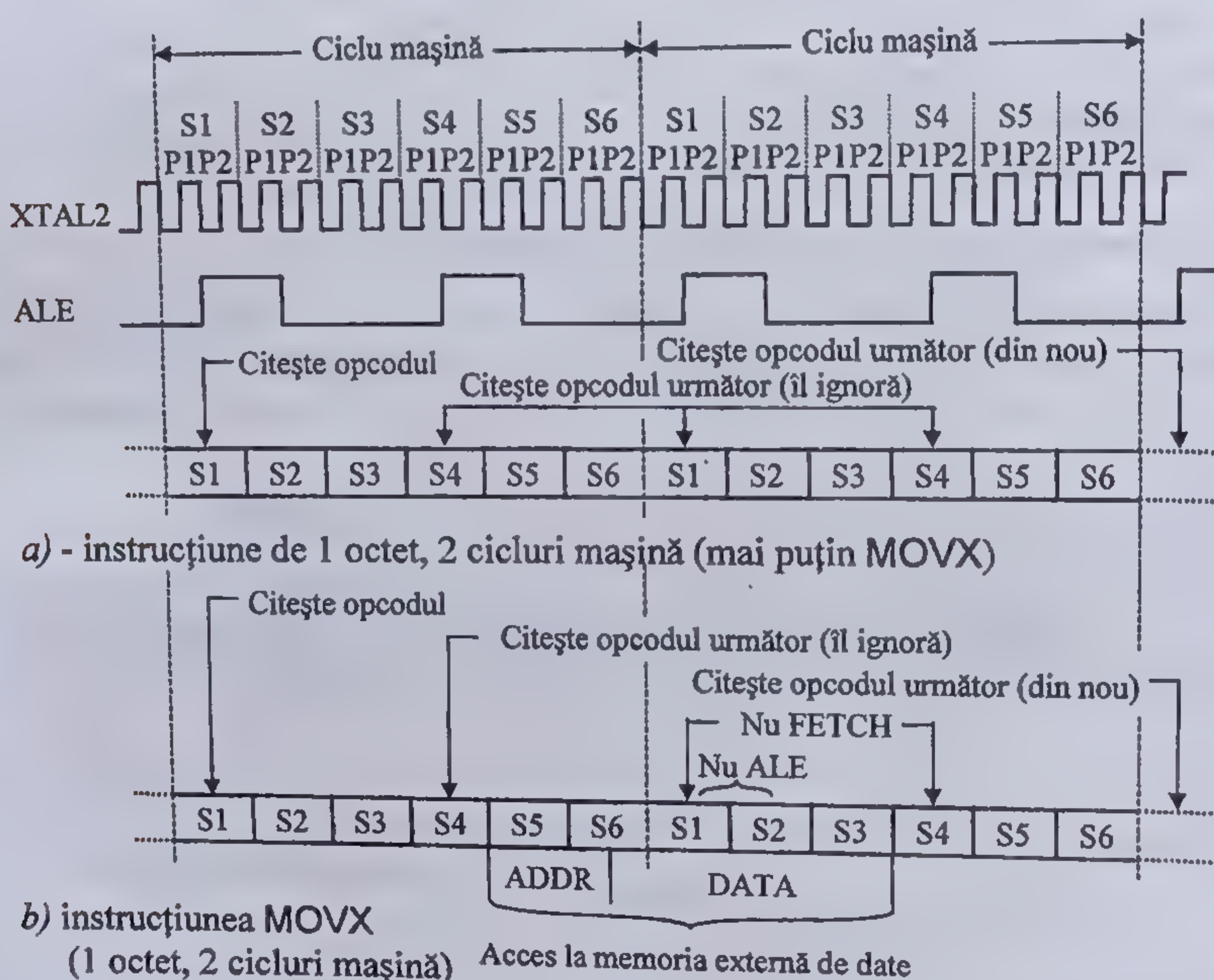


Fig.6.10. Extragerea și execuția instrucțiunilor de 2 cicluri mașină

în ciclul 2 nu se execută operațiile fetch și nu se generează ALE.

În fig.6.11 se prezintă secvențierea explicită a operațiilor în cazul execuției instrucțiunilor din memoria externă. Cronogramele de semnal în absența unor instrucțiuni MOVX sunt prezentate în fig.6.11a. Execuția unei instrucțiuni MOVX modifică secvențierea operațiilor ca în fig.6.11b. Eșantionarea datelor la citirea memoriei RAM externe se face în decît un acces la memoria externă de date durează de două ori mai mult

Atunci când CPU execută instrucțiuni din memoria program internă, $\overline{\text{PSEN}}$ nu se mai activează, iar adresa de memorie nu se mai emite. Semnalul ALE continuă să se activeze de

două ori pe ciclu mașină și poate fi folosit ca semnal de tact. Numai în cazul execuției unei instrucțiuni MOVX este omis cel de-al doilea impuls ALE.

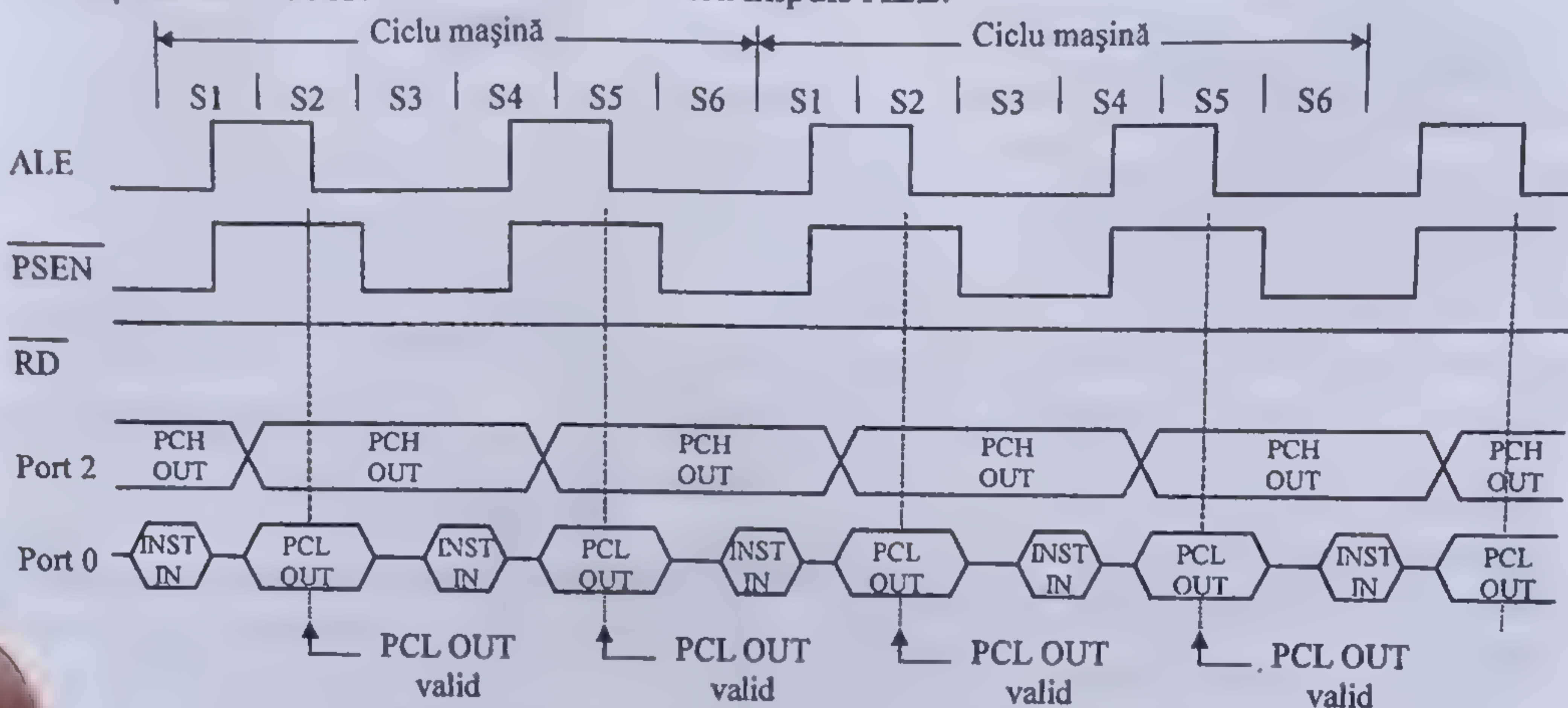


Fig.6.11a. Execuția instrucțiunilor din memoria program externă (fără MOVX)

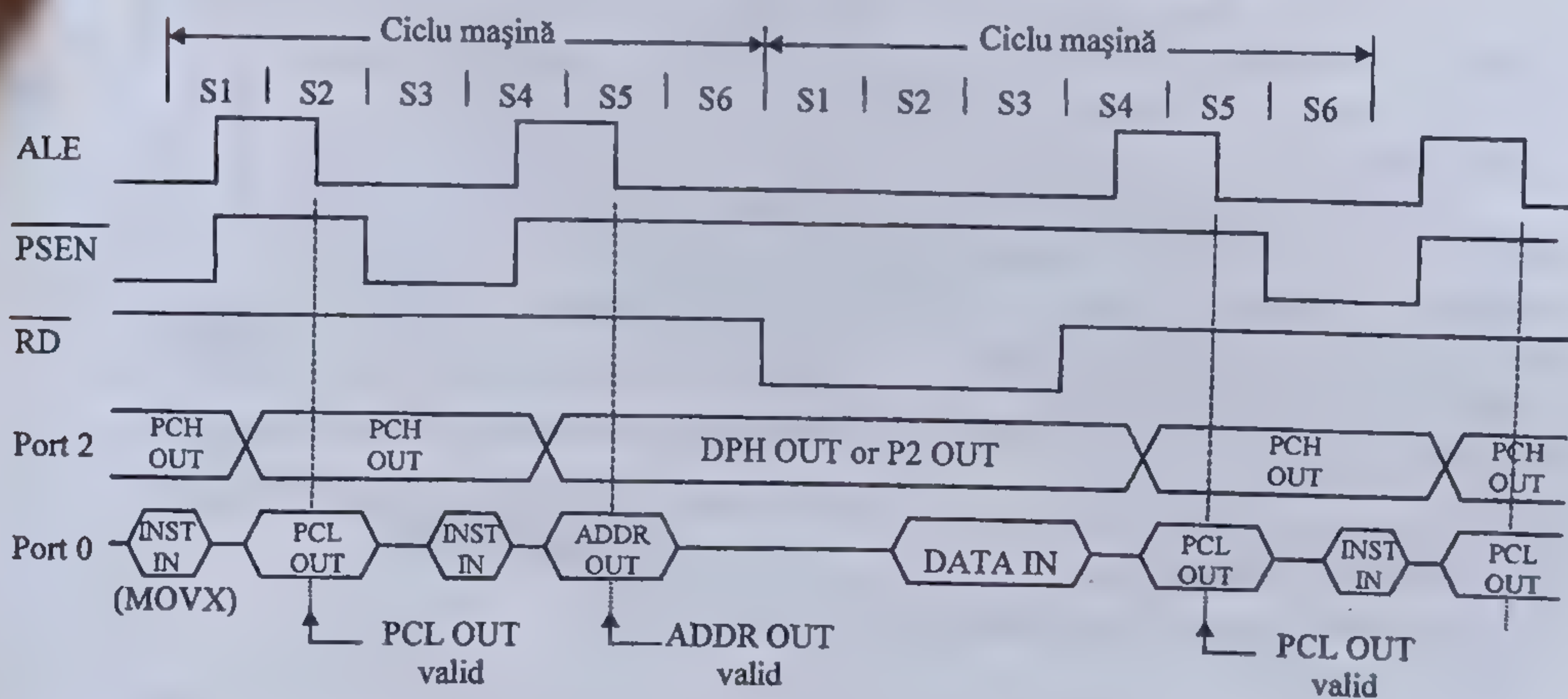


Fig.6.11b. Execuția unei instrucțiuni MOVX din memoria program externă

Acest ultim exemplu evidențiază posibilitatea dispunerii programului și în memoria RAM externă (fig.6.12), fapt avantajos pentru sistemele de dezvoltare. Deoarece în memoria externă comună coexistă ambele tipuri de informații, este necesar ca citirea să poată fi realizată fie cu semnalul $\overline{\text{PSEN}}$, fie cu $\overline{\text{RD}}$, în conformitate cu relația:

$$\overline{\text{MEMRD}} = \overline{\text{PSEN}} + \overline{\text{RD}} = \overline{\text{PSEN}} \cdot \overline{\text{RD}}.$$

Memoria utilizată trebuie să aibă timpul de acces corespunzător situației celei mai dezavantajoase: citire ca memorie program.

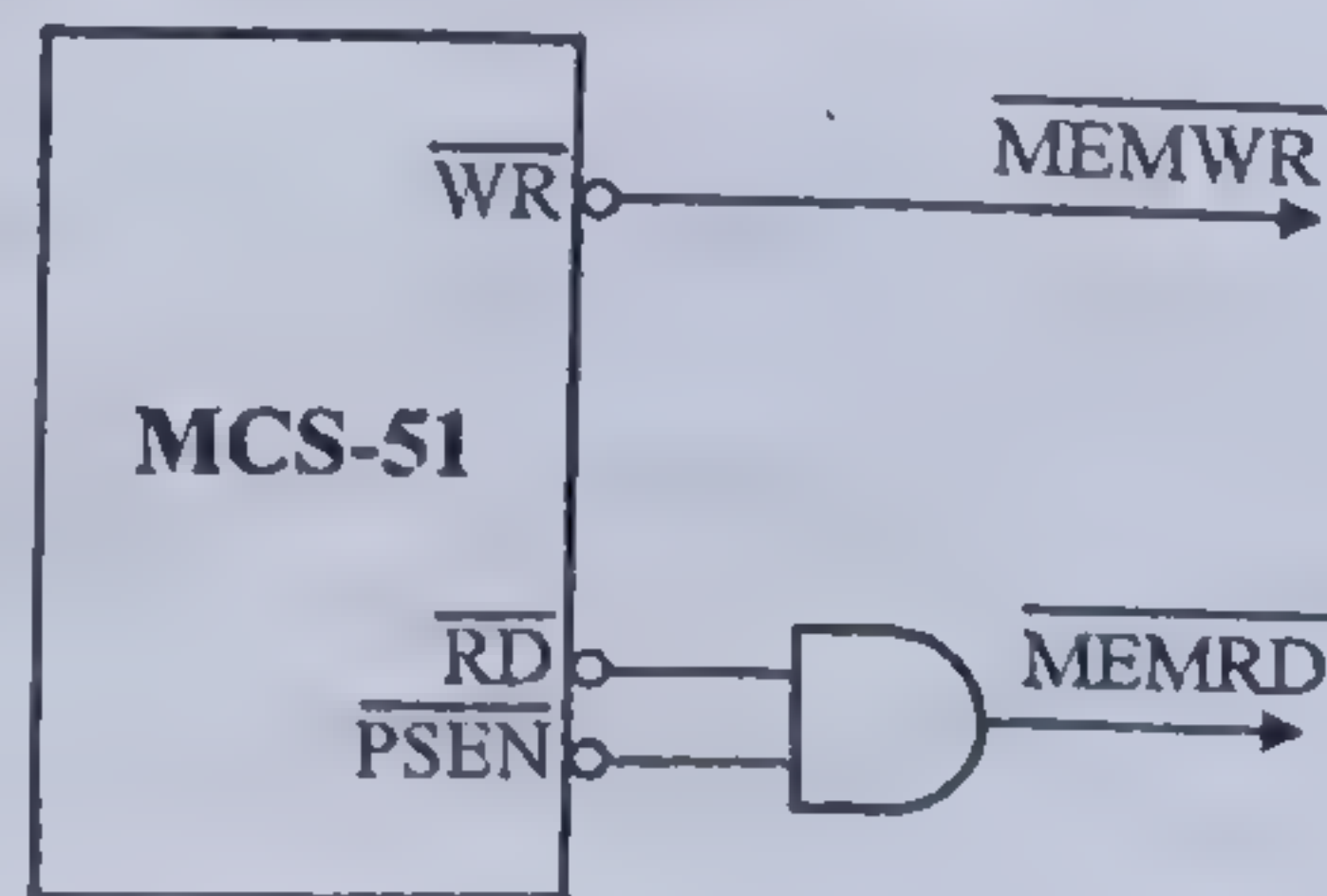
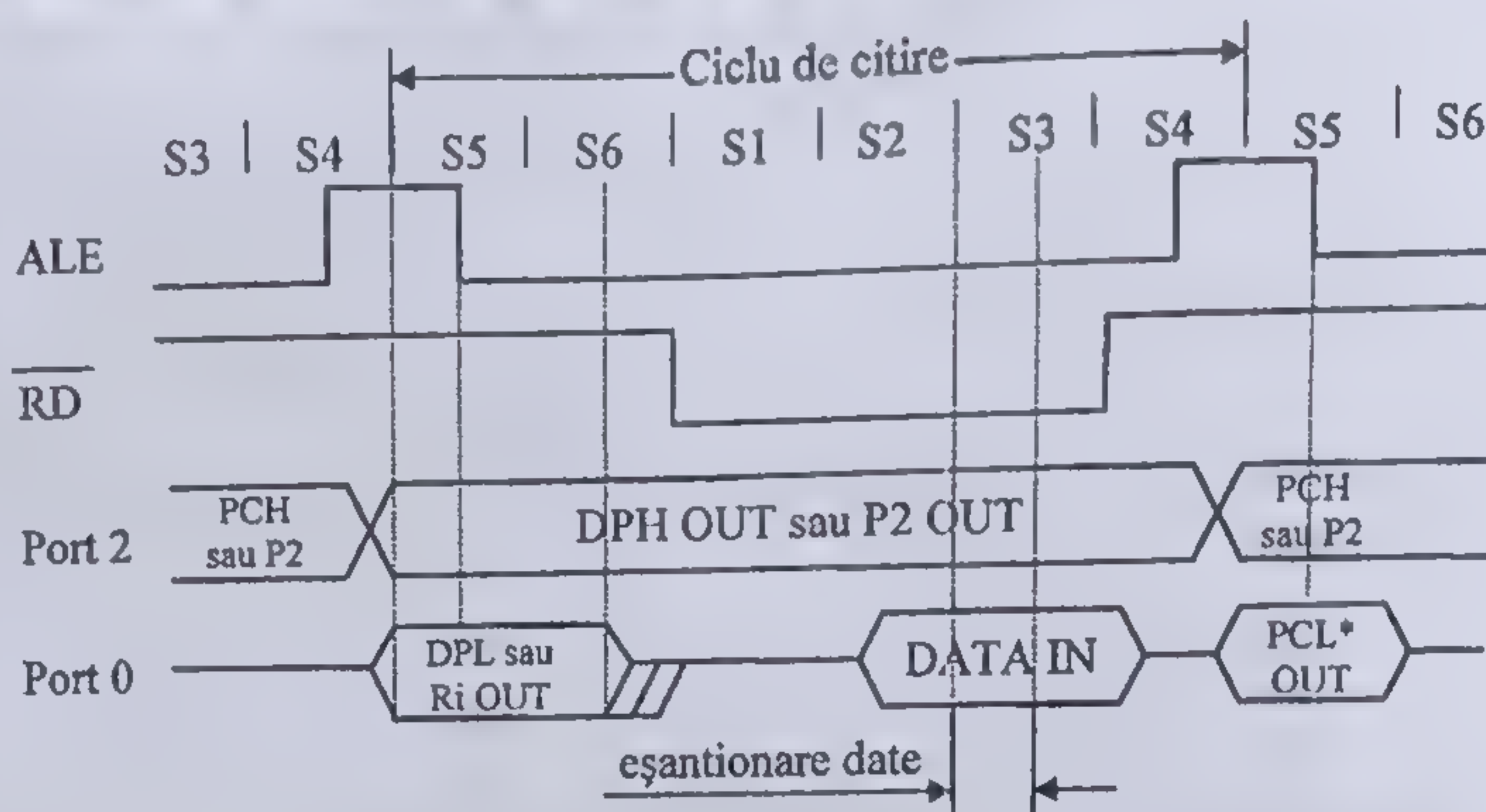


Fig.6.12. Combinarea spațiului extern de memorie program și de date

6.3.2.3. Citirea și înscirerea datelor în memoria externă de date

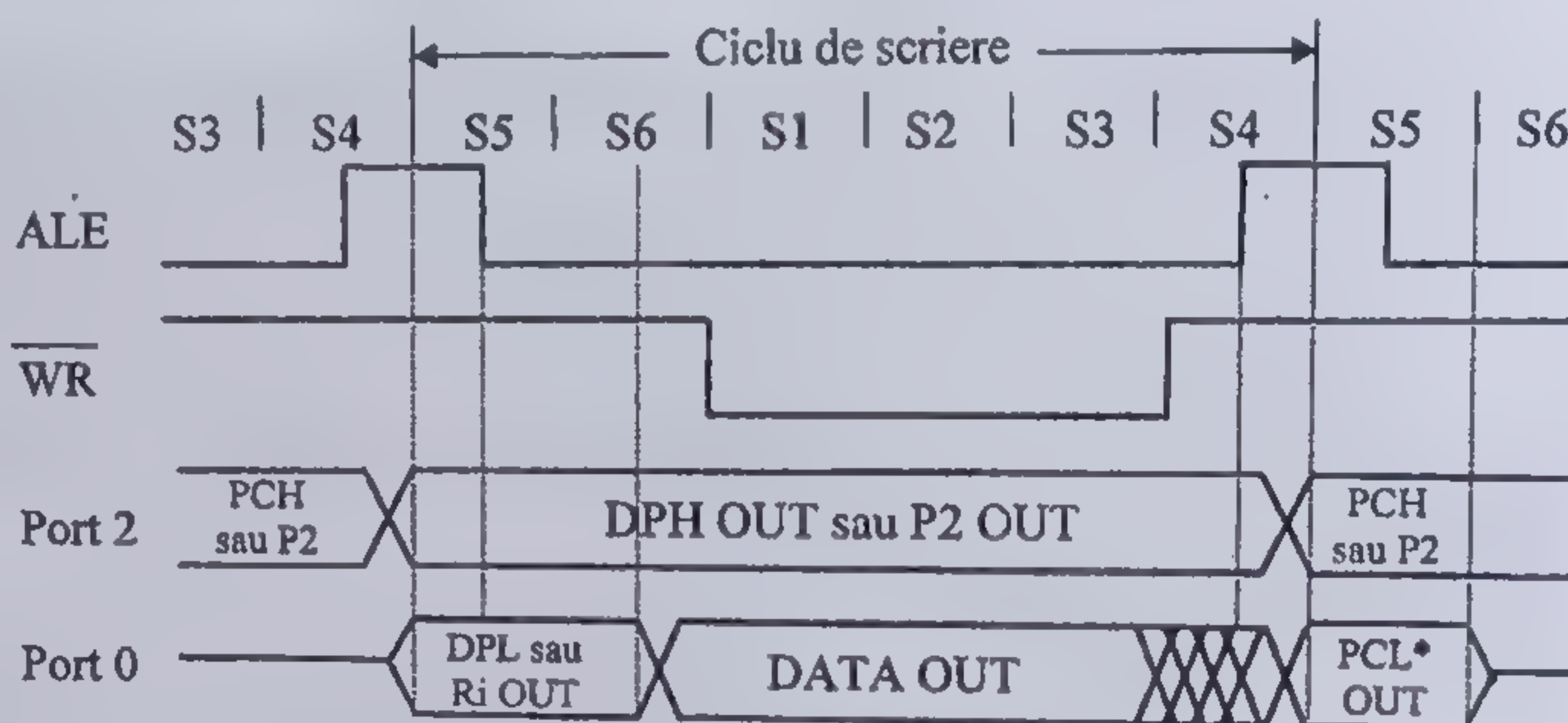
Aceasta decurge în conformitate cu diagramele din fig.6.13. În fig.6.13a este prezentată secvențierea operației de citire din memoria externă de date. Portul 2 furnizează pe întreaga durată a ciclului conținutul DPH, iar dacă nu se folosește o adresă de 16 biți, conținutul SFR al Portului 2 (P2). Acest fapt facilitează organizarea paginată a memoriei externe. Portul 0 conține DPL sau Ri ($i=0,1$) - dacă adresa este de 8 biți. Datele depuse de memorie sunt eșantionate în S3P1. Dacă și memoria program este externă, atunci cele două porturi vor conține PC (PCH și PCL).

În cazul ciclului de înscirere a informației în memoria de date externă (fig.6.13b), după depunerea adresei de 16 biți (DPH și DPL) sau de 8 biți (Ri), pe liniile Portului 0 se depun datele, începând cu S6P2. Apoi, între S1 și S2 se activează \overline{WR} , după care se transmite PCL și respectiv PCH.



PCL* OUT - dacă instrucțiunea se execută din memoria programe externă

Fig.6.13a. Secvențierea operațiilor la citirea memoriei de date



PCL* OUT - dacă instrucțiunea se execută din memoria program externă

Fig.6.13b. Secvențierea operațiilor la înscirerea memoriei de date

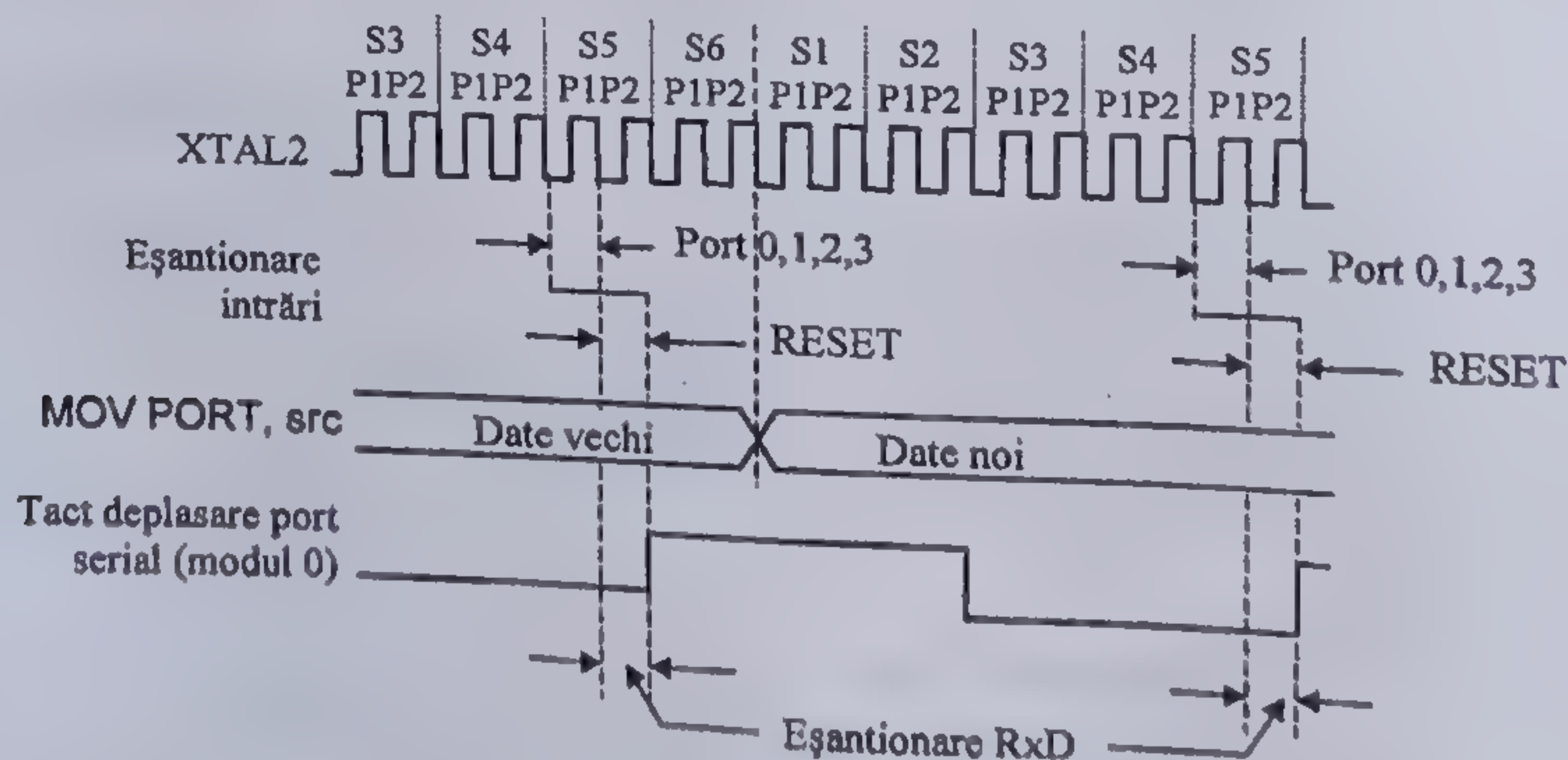


Fig.6.14. Secvențierea operațiilor de transfer prin porturi

La transferul datelor prin cele 4 porturi paralele (Port 0÷3) și prin portul serial, secvențierea se realizează conform diagramei din fig.6.14. Data înscrisă într-un port printr-o instrucțiune MOV PORT,src (source) ajunge în bistabilele SFR în S6P2 și apare la pini în starea S1P1 a următorului ciclu mașină. Data dintr-un port este eșantionată în starea S5P1 a ultimului ciclu mașină al unei instrucțiuni de citire port.

Intrarea RxD a portului serial este eșantionată în S5P2, înaintea trecerii în "1" a tactului de deplasare la funcționarea în modul 0 (registru de deplasare). Tot în S5P2 microcontrolerul eșantionează și intrarea de resetare. Detalii privind secvențierea operațiilor la transferul serial al datelor vor fi prezentate în §6.3.4.2.

6.3.3. Porturile de intrare-ieșire. Structură și operare

Toate porturile familiei de bază, Port 0÷3 (vezi fig.6.2), sunt bidirecționale. Fiecare port posedă un registru cu memorare (SFR-urile P0÷P3), un amplificator (driver) de ieșire și un tampon de intrare. Driverile de ieșire ale Porturilor 0 și 2 și tamponul de intrare al Portului 0 sunt utilizate, după cum s-a văzut în paragraful precedent, pentru accesul la memoria externă. Portul 0 transmite în acest caz octetul inferior al adresei, multiplexat în timp cu octetul de date în ciclurile de citire sau înscrisiere. Portul 2 furnizează octetul superior al adresei de 16 biți. În caz contrar, pe liniile acestuia este depus conținutul SFR P2.

Toate liniile Portului 3, iar în cazul MCU 8052 și două linii ale Portului 1, sunt multifuncționale. Acestea servesc nu numai ca linii I/E de uz general, ci și pentru funcții alternative speciale, așa cum s-a arătat în §6.2. În tab.6.6 sunt prezentate detaliat aceste funcții speciale, împreună cu liniile aferente ale Porturilor 1 și 3. Funcțiile alternative pot fi activate numai dacă biții corespunzători ai SFR pentru P3 și respectiv pentru P1 (P1.0, P1.1) sunt poziționați în "1". În caz contrar, biții respectivi trebuie fixați la "0" logic.

6.3.3.1. Configurația porturilor I/E

În fig.6.15 se prezintă schemele funcționale la nivel de bit pentru porturile paralele de I/E. Astfel, un bit din SFR-ul portului este reprezentat printr-un bistabil de tip D. Acesta reține valoarea de pe magistrala internă la o comandă "scrie în SFR", primită de la CPU. Ieșirea Q a bistabilului este plasată pe magistrala internă, ca răspuns a unei comenzi "citește SFR", transmisă de CPU. De asemenea, o comandă "citește pinul" va transmite starea pinului portului adresat pe magistrala internă. Semnalele de comandă "citire SFR" și "citire pin" sunt activate de instrucțiuni diferite, așa cum se va arăta în §6.3.3.2.

Așa cum se observă în fig.6.15a și 6.15c, ieșirile Porturilor 0 și 2 pot fi conectate la magistralele Adrese/Date, respectiv Adrese, printr-un semnal intern de comandă, pentru accesul la memoria externă. Pe durata acestuia, P2 rămâne neschimbat, dar în P0 se încarcă FFh.

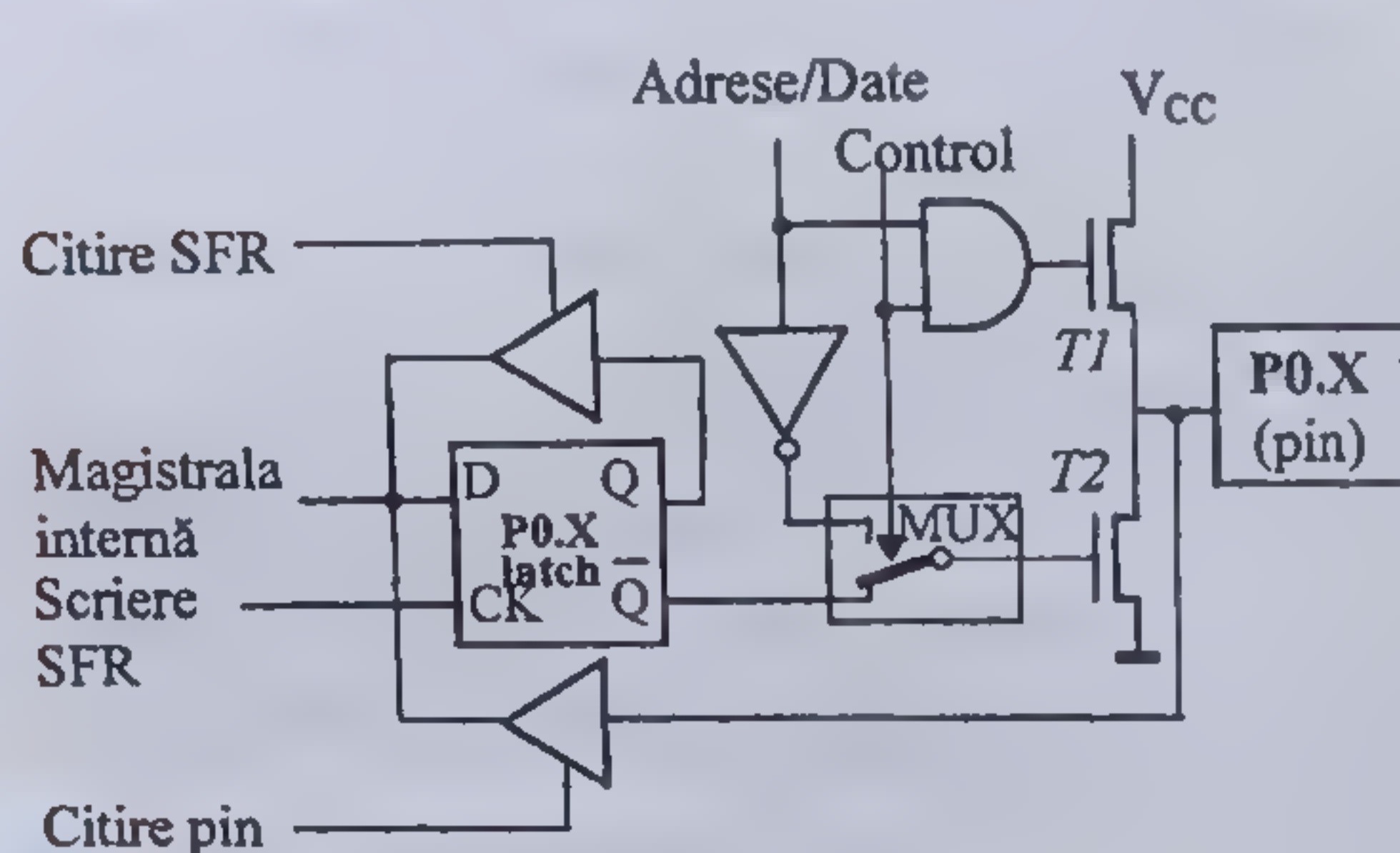
Denumire pin	Funcția alternativă
*P1.0	T2 - Intrare externă pentru Timerul 2
*P1.1	T2EX - Intrare pentru declanșarea funcționării Timerului 2
P3.0	RxD - Intrare Port serial
P3.1	TxD - Ieșire Port serial
P3.2	INT0 - Întrerupere externă 0
P3.3	INT1 - Întrerupere externă 1
P3.4	T0 - Intrare externă pentru Timerul 0
P3.5	T1 - Intrare externă pentru Timerul 1
P3.6	WR - Semnal de înscrisiere pentru memoria externă de date
P3.7	RD - Semnal de citire pentru memoria externă de date

*P1.0 și P1.1 au aceste funcții numai pentru 8052

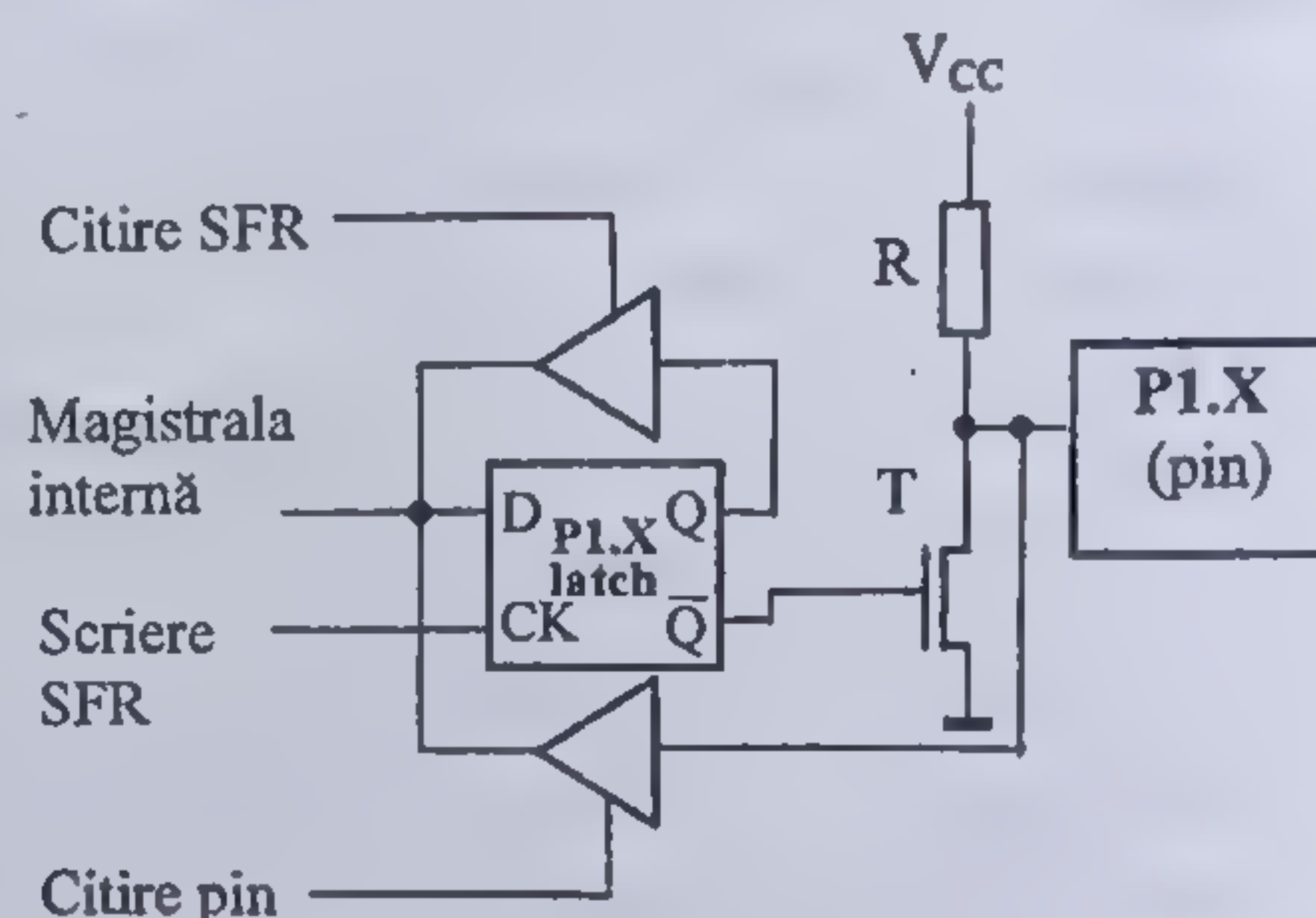
Tab.6.6. Funcțiile alternative ale pinilor

Portul 3 poate transmite funcțiile alternative (v.tab.6.6) dacă SFR-ul său are biții corespunzători poziționați pe "1" logic, ieșirile fiind controlate de semnalul "funcție de ieșire alternativă". Starea pinilor P3.X este disponibilă la "funcțiile de intrare alternativă" aferente.

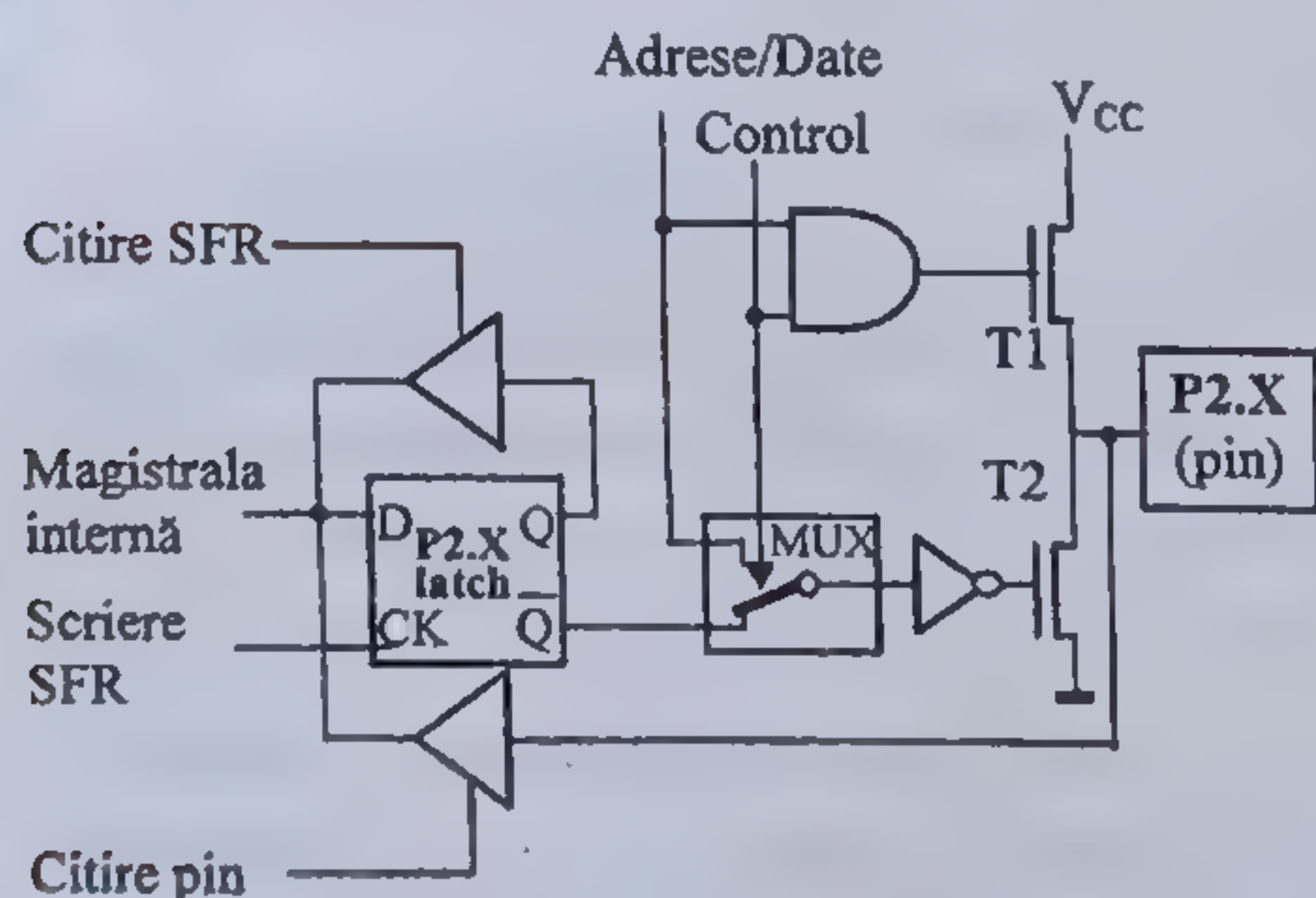
Ca porturi de intrare-ieșire, liniile pot fi utilizate independent fie ca intrări, fie ca ieșiri. Evident, dacă Porturile 0 și 2 sunt folosite ca magistrale de adrese și de date, atunci acestea nu se pot utiliza și ca porturi I/E. Porturile 1, 2 și 3 au rezistoare interne de fixare ("pull-up"), în timp ce Portul 0 este de tipul cu drenă în gol. Tranzistorul T1 de la ieșirile Portului 0 este utilizat numai când acesta emite un "1" logic, în timpul accesului la memoria externă. În celelalte situații, tranzistorul de fixare este blocat. În consecință, liniile Portului 0 utilizate ca ieșiri sunt cu drenă în gol și necesită rezistoare externe conectate la V_{CC} . Un "0" logic în SFR deschide tranzistorul T2 și semnalul "0" se transmite la ieșire. Înscrierea unui "1" în SFR-ul Portului 0 menține ambele tranzistoare de ieșire blocate, ceea ce asigură flotarea pinilor. În aceste condiții, liniile acestui port pot fi folosite ca intrări de înaltă impedanță. Ca urmare, Portul 0 se consideră un port bidirecțional "adevărat", deoarece atunci când este configurat ca port de intrare, bufferele de ieșire se află în HZ.



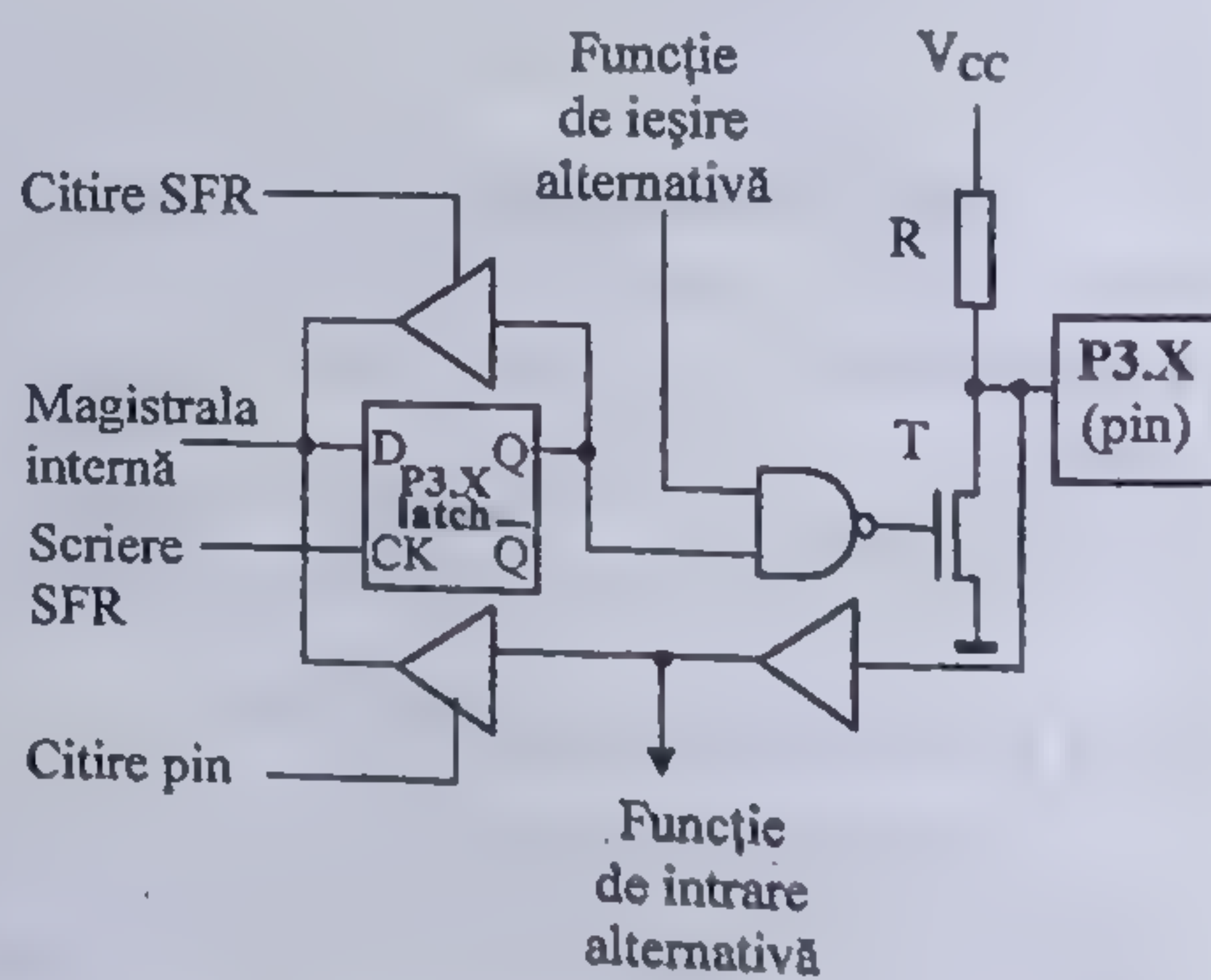
a) Structura internă a Portului 0



b) Structura internă a Portului 1



c) Structura internă a Portului 2



d) Structura internă a Portului 3

Fig.6.15. Configurația I/E a porturilor familiei MCS-51

Datorită rezistoarelor interne de fixare, formate dintr-un complex de tranzistoare FET, Porturile 1, 2 și 3 sunt denumite porturi cvasi-bidirecționale. Dacă sunt configurate ca intrări, aceste rezistoare de fixare au valoare mare și vor fi surse de curent când în exterior se aplică un

“0” logic. Ca și în cazul Portului 0, liniile acestor porturi pot fi configurate independent, fie ca intrări, fie ca ieșiri:

- o linie poate fi utilizată ca intrare dacă bitul respectiv din SFR este pe “1” logic;
- ca ieșire, linia este comandată prin intermediul bitului corespunzător din SFR. Un “1” în SFR blochează tranzistorul T al etajului final, în timp ce un “0” în SFR determină deschiderea tranzistorului de ieșire. În primul caz ieșirea este în “1” logic, iar în al doilea caz în “0” logic.

În cazul funcționării ca magistrală externă, Portul 0 poate comanda 8 sarcini LS TTL. Ca port I/E, sunt necesare rezistoare de fixare externe pentru a comanda fiecare intrare. Porturile 1, 2 și 3 pot comanda 4 sarcini LS TTL.

6.3.3.2. Facilitatea “citire-modificare-scriere” a porturilor

Operația de citire dintr-un port poate fi realizată fie cu instrucțiuni care citesc informația direct de la pini, fie cu instrucțiuni care citesc SFR-ul portului. Prima categorie poate conduce la interpretarea incorectă a datelor citite. De exemplu, dacă un pin este utilizat pentru comanda unui tranzistor prin bază, înscrierea unui bit “1” în port determină deschiderea tranzistorului. Citirea directă a pinului va fi interpretată în acest caz ca “0” logic (tensiunea bază-emitor a tranzistorului deschis) și nu valoarea corectă, “1” logic. Pentru evitarea acestor erori se folosesc instrucțiuni de citire din SFR-ul asociat portului, care au facilitatea de a citi informația, eventual de a o modifica și apoi o reînscriu în latch. Astfel de instrucțiuni sunt denumite instrucțiuni de *citire-modificare-însciere*. Când operandul destinație este un port sau bitul unui port, aceste instrucțiuni citesc SFR-ul și nu pinul corespunzător. În tab.6.7 sunt prezentate instrucțiunile de citire-modificare-însciere, împreună cu semnificația lor și exemple de utilizare.

Instrucțiunea	Semnificația	Exemple
ANL	SI logic	ANL P1,A; ANL P3,#40h
ORL	SAU logic	ORL P2, A; ORL P1#55h
XRL	SAU EXCLUSIV (suma modulo 2)	XRL P3,A; XRL P0,# 0FFh
JBC	Salt dacă bitul este 1, apoi șterge bitul	JBC P1.1,ADR1
CPL	Complementarea bitului	CPL P3.2
INC	Incrementare	INC P2
DEC	Decrementare	DEC P0
DJNZ	Decrementează și salt dacă nu este zero	DJNZ P3, ADR 2
MOV PX.Y,C	Transferă conținutul bitului CY în bitul Y al portului X	MOV P1.1,C
CLR PX.Y	Resetează bitul Y al portului X	CLR P0.7
SETB PX.Y	Setează bitul Y din portul X	SETB P3.2

Tab.6.7. Instrucțiuni de tip “citire-modificare-însciere”

La prima vedere s-ar părea că ultimele trei instrucțiuni nu sunt de tipul “citire-modificare-însciere”, însă ele sunt de acest tip deoarece efectuează următoarele operații: citirea celor 8 biți ai portului, modificarea bitului adresat și apoi înscierea noului octet în SFR.

6.3.4. Funcții periferice integrate

Familia MCS-51 dispune de resurse integrate pe cip care asigură noi facilități, specifice microprocesoarelor orientate pe control, cum ar fi:

- urmărirea evoluției evenimentelor în timp real;
- contorizarea tranzițiilor semnalelor;
- măsurarea precisă a duratei impulsurilor;
- comunicarea cu alte sisteme sau cu operatorul uman;
- monitorizarea evenimentelor asincrone externe.

Toate cele trei timere pot fi programate să opereze în două moduri de bază: fie ca *numărătoare de evenimente*, fie ca *temporizatoare*.

În modul *temporizator*, numărătorul unui timer este incrementat la fiecare ciclu mașină. Dina acest punct de vedere temporizatorul poate fi asimilat cu un contor de cicluri mașină. Deoarece un ciclu mașină constă din 12 perioade ale oscilatorului de tact, rata de incrementare este egală cu $1/12$ din frecvența oscilatorului intern al microcontrolerului.

În modul *numărător*, contorul timerului este incrementat la fiecare tranziție $1 \rightarrow 0$ a semnalelor externe de la pinii corespunzători (vezi tab.6.6): T0, T1 sau T2 (ultimul numai la seria 8052). Semnalele la pini sunt eșantionate pe durata fazei S5P2 a fiecărui ciclu mașină. Dacă în momentul testării semnalul este "0" logic, iar în ciclul anterior a fost "1" logic, numărătorul este incrementat. Noua valoare apare în registrele SFR în faza S3P1 a ciclului următor celui în care a fost detectată tranziția. Deoarece recunoașterea unei tranziții $1 \rightarrow 0$ necesită două cicluri mașină (24 perioade oscilator), rata maximă de contorizare este $1/24$ din frecvența oscilatorului. Nu există nici o restricție privitor la parametrii semnalului extern, dar pentru a avea siguranța unei eșantionări corecte, nivelul acestuia trebuie menținut până la terminarea ciclului mașină.

Pentru fiecare din cele două moduri de bază definite mai sus, în funcție de dimensiunea și de funcționalitatea contorului cu incrementare, pentru Timerul 0 și Timerul 1 se pot stabili patru moduri de operare selectabile prin TMOD. În mod similar, pentru Timerul 2 există trei moduri de operare, selectabile prin T2CON.

Timerele 0 și 1

Selectarea modurilor de bază, numărător sau temporizator, se realizează prin biții C/\bar{T} din registrul TMOD (fig.6.16): dacă sunt pe "0", cele două timere funcționează ca "temporizator" cu intrare de la tactul intern, iar dacă sunt pe "1" funcționează ca "numărător", cu intrarea de la pinii T0, respectiv T1.

Funcționarea (incrementarea) fiecărui contor poate fi controlată direct prin program (SW), cu biții TRx (x = 0 sau 1) din registrul TCON (fig.6.17), dar și prin semnal (HW), validat de biții GATE din TMOD. Astfel, dacă bitul corespunzător GATE este poziționat pe "1", funcționarea Timerului "x" este validată/inhibată de valoarea logică "1", respectiv "0" aplicată pe pinul \overline{INTx} , ceea ce facilitează măsurarea lățimii impulsurilor. Dacă bitul GATE este poziționat pe "0" logic, atunci funcționarea Timerului "x" poate fi pornită/oprită numai prin program, prin setarea/resetarea bitului TRx.

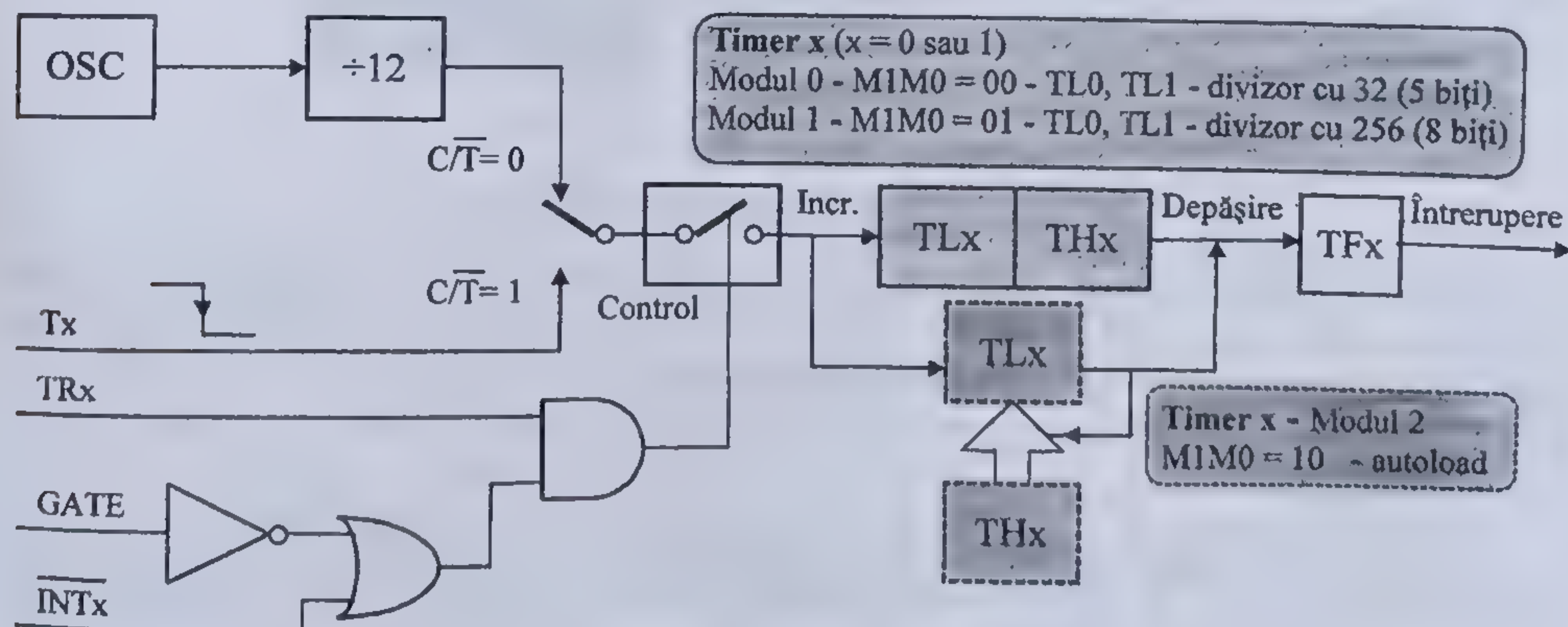


Fig.6.18. Funcționarea Timerelor 0 și 1 în modurile 0, 1 și 2

Timerele 0 și 1 au patru moduri de operare, selectabile prin biții pereche (M1,M0) din TMOD. Modurile 0, 1 și 2 sunt identice la ambele timerc, însă modul 3 este diferit.

Modul 0 corespunde funcționării ca numărător de 8 biți cu divizor de 5 biți, compatibil cu funcționarea timerului de la MCS-48. În acest mod, registrele Timerelor 0 și 1 sunt configurate ca registre de 13 biți: THx de 8 biți, iar TLx de 5 biți. Cei mai semnificativi trei biți din TLx sunt nedeterminați și trebuie ignorați. La depășirea capacității contorului de 8 biți THx, la tranziția FFh→00h, se setează fanionul de întrerupere TFX.

Modul 1 este asemănător modului 0, dar contoarele timerelor sunt de 16 biți: THx-TLx (v. fig.6.16 și 6.17).

Modul 2 configurează registrele TLx ca numărătoare cu reîncărcare automată, cu valoarea preprogramată în THx (fig.6.18). Reîncărcarea se realizează la depășire, fapt semnalizat și de fanioanele TFX din TCON. Registrele THx nu sunt afectate de reîncărcare.

Modul 3 poate fi utilizat numai de Timerul 0, în timp ce Timerul 1 este oprit. Registrele TL0 și TH0 formează două numărătoare/temporizatoare independente de 8 biți, așa cum se arată în fig.6.19. Canalul realizat cu TL0 folosește biții de control de la Timerul 0: C/T, GATE, TR0, $\overline{\text{INT0}}$ și TF0. Canalul realizat cu TH0 poate lucra doar ca temporizator (numărător de cicluri mașină) și folosește semnalele TR1 și TF1 de la Timerul 1. La depășire, ambele canale pot lansa cereri de întrerupere: TL0 prin TF0, iar TH0 controlează întreruperea Timerului 1. În tot acest timp, Timerul 1 poate fi încă utilizat în orice aplicație care nu necesită întreruperi, ca numărător comandat prin pinul T1 sau ca temporizator în oricare din modurile 0, 1 sau 2. Pentru oprire/pornire este folosită intrarea/ieșirea în/din modul 3. De asemenea, mai poate fi folosit de portul serial ca *generator de viteză de transfer* ("baud rate generator").

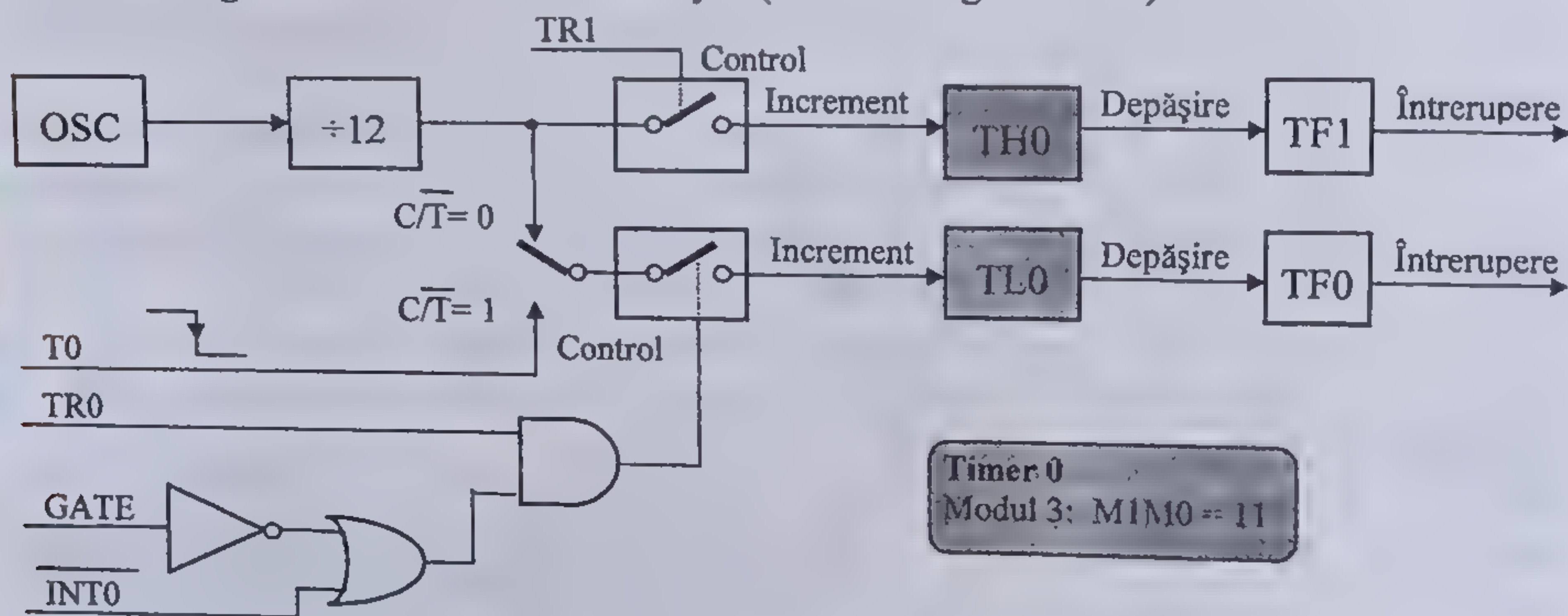


Fig.6.19. Funcționarea Timerului 0 în modul 3

Astfel, modul 3 este avantajos în aplicațiile care necesită un numărător sau un temporizator de 8 biți suplimentar. Deci 8051 poate fi considerat în acest mod ca având trei numărătoare/temporizatoare, iar 8052 un număr de patru.

Timerul 2

După cum s-a menționat, Timerul 2 există numai la seria 8052 și are asociate două semnale externe: T2 și T2Ex (vezi tab.6.6). Selectarea funcționării ca numărător sau temporizator se face prin bitul C/T2 din registrul T2CON (fig.6.20).

Cu ajutorul biților RCLK, TCLK, CP/RL2 și TR2 din T2CON pot fi selectate prin program cele trei moduri de operare ale Timerului 2, așa cum se arată în tab.6.8. În modul "captură" există două opțiuni care pot fi selectate prin bitul EXEN2 din T2CON. Dacă EXEN2=0, atunci

RCLK+TCLK	CP/RL2	TR2	Mod de operare
0	0	1	Reîncărcare pe 16 biți
0	1	1	Captură pe 16 biți
1	x	1	Generator viteză comunicație serială
x	x	0	Deconectat

Tab.6.8. Modurile de operare ale Timerului 2

Timerul 2 lucrează ca un numărător/temporizator de 16 biți obișnuit, care își setează fanionul TF2 la depășire și poate lansa o cerere de întrerupere. Dacă EXEN2=1, Timerul 2 va avea o facilitare suplimentară: o tranziție 1→0 pe intrarea T2EX determină "captarea" conținutului registrelor TL2 și TH2 în registrele RCAP2L și respectiv RCAP2H.

		(msb)							(lsb)		
		TF2	EXF2	RCLK	TCLK	EXEN	TR2	C/T2	CP/RL2	(C8h)	
Simbol	Poziție	Nume și semnificație									
TF2	T2CON.7	Fanion de depășire Timer 2 (Timer 2 overflow Flag). Este setat când Timerul 2 realizează o depășire și trebuie resetat SW. TF2 nu va fi setat dacă RCLK=1 sau TCLK=1.									
EXF2	T2CON.6	Fanion extern (Timer 2 External Flag). Este setat când EXEN2=1 și o tranziție negativă pe T2EX determină o captură/reîncărcare. Dacă este activată întreruperea Timerului 2, atunci EXF2=1 va cauza o cerere de întrerupere. EXF2 trebuie resetat SW.									
RCLK	T2CON.5	Fanion selecție tact de recepție (Receive Clock Flag) pentru portul serial în modurile 1 și 3: RCLK=1 - se folosesc impulsurile de depășire de la Timerul 2; RCLK=0 - se folosesc impulsurile de depășire de la Timerul 1.									
TCLK	T2CON.4	Fanion selecție tact transmisie (Transmit Clock flag) pentru portul serial în modurile 1 și 3: TCLK=1 - de la Timerul 2; TCLK=0 - de la Timerul 1.									
EXEN2	T2CON.3	Fanion pentru activarea captării la Timerul 2 (Timer 2 external enable flag). Dacă Timerul 2 nu este folosit pentru portul serial și EXEN2=1, o tranziție negativă pe T2EX determină o captură/reîncărcare a Timerului 2; EXEN2=0 determină ignorarea evenimentelor pe T2EX.									
TR2	T2CON.2	Bit de control a funcționării Timerului 2 (Timer 2 Run control bit). TR2=1 - start Timer 2; TR2=0 - stop Timer 2.									
C/T2	T2CON.1	Bit selecție mod de lucru Timer 2. 0 = Temporizator (OSC/12) 1 = Numărător de evenimente externe (front descrescător)									
CP/RL2	T2CON.0	Fanion captură/reîncărcare (Capture/Reload flag). Când este setat se realizează captura la o tranziție negativă pe T2EX, dacă EXEN2=1. Dacă este 0, se realizează reîncărcarea automată a Timerului 2 la depășire sau la o tranziție negativă pe T2EX, când EXEN2=1. Dacă RCLK=1 sau TCLK=1, acest bit este ignorat și timerul este forțat să se reîncarce automat la depășire.									

Fig.6.20. T2CON - registrul de comandă al Timerului 2

În fig.6.21 este prezentată o schemă explicativă a funcționării Timerului 2 în modul "captură". Odată cu captarea informației Timerului 2, se setează și fanionul EXF2 din T2CON și se poate lansa o cerere de întrerupere spre CPU.

În modul "reîncărcare", ilustrat în fig.6.22, există de asemenea două opțiuni, selectabile tot prin EXEN2 din T2CON. Dacă EXEN2=0, atunci depășirea Timerului 2 setează TF2 și determină reîncărcarea registrelor TL2 și TH2 cu valoarea de 16 biți presetată în registrele RCAP2L, respectiv RCAP2H. Dacă EXEN2=1, Timerul 2 funcționează ca mai înainte, dar cu o nouă facilitate: la o tranziție 1→0 pe T2EX, va fi realizată reîncărcarea pe 16 biți și EXF2 va fi setat.

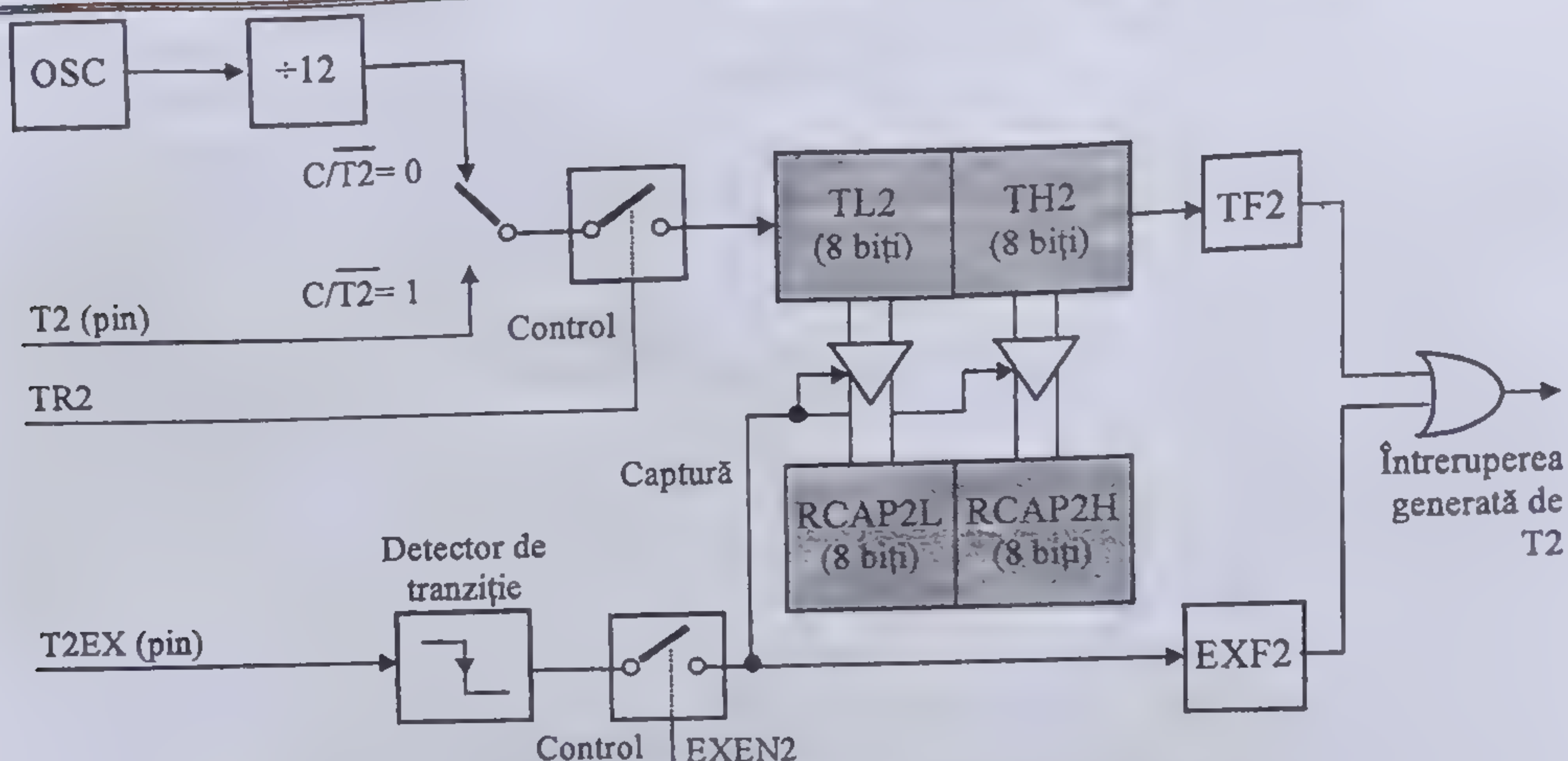


Fig.6.21. Timerul 2 - funcționarea în modul "captură"

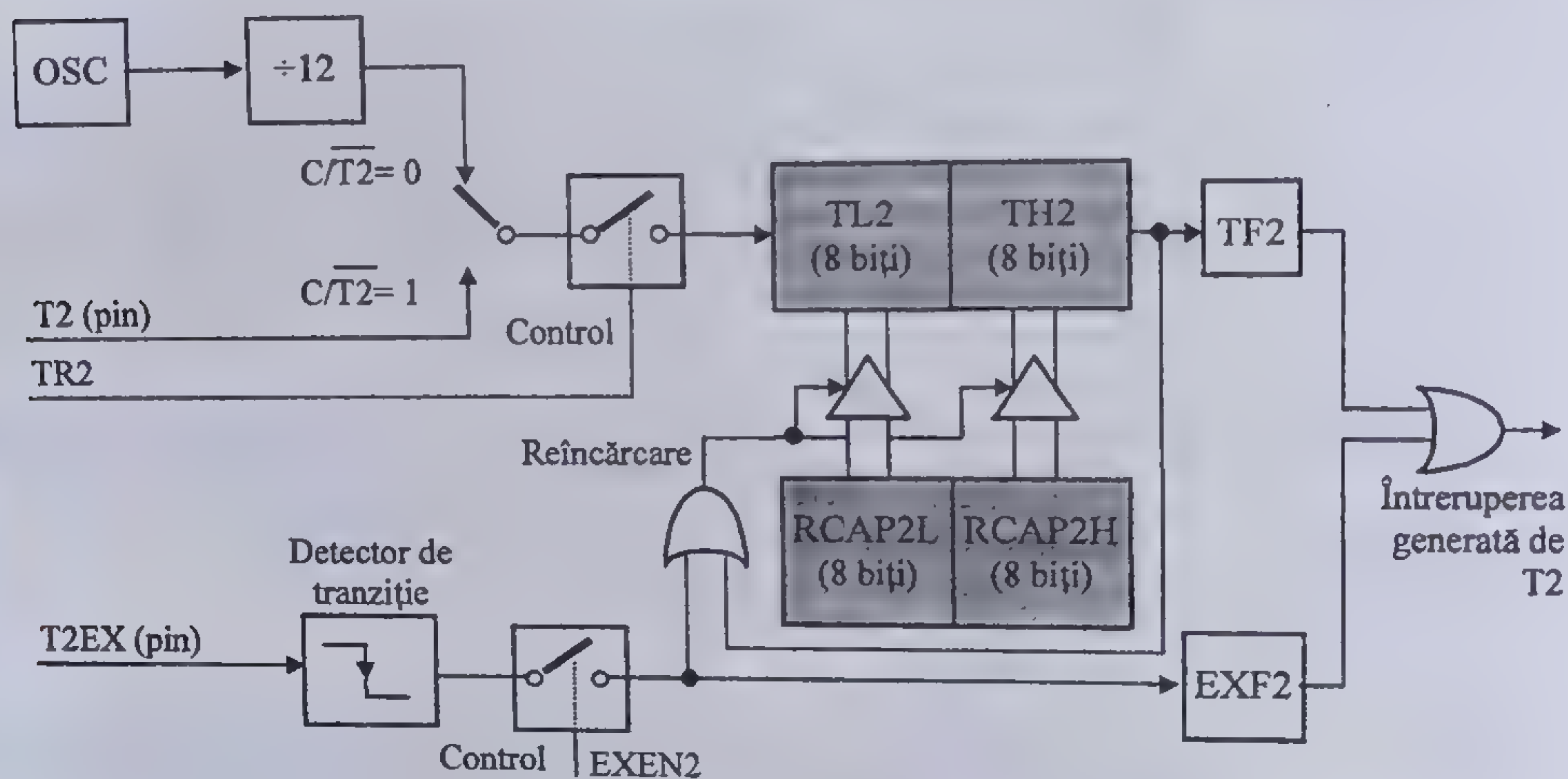


Fig.6.22. Timerul 2 - funcționarea în modul "reîncărcare"

Modul "generator de viteză de comunicație serială" (tab.6.8) este selectat prin $RCLK=1$ și/sau $TCLK=1$. Amănunte asupra acestui mod de funcționare vor fi date în paragraful următor.

6.3.4.2. Interfața serială - moduri de operare

Portul serial este de tip "full duplex", permițând transmitia și recepția simultană a datelor. De asemenea, recepția este bufferizată, ceea ce asigură începerea recepției unui octet înainte ca precedentul să fi fost citit din registrul de recepție. Totuși, dacă citirea primului octet nu se efectuează în timpul recepției celui de-al doilea octet, primul octet se va pierde. Modurile de operare ale portului serial pot fi controlate prin intermediul registrului SCON (fig.6.23).

De asemenea, prin SCON se pot controla întreruperile portului, atât la transmisie (TI) cât și la recepție (RI), precum și cel de-al 9-lea bit (TB8, RB8) la funcționarea UART cu 9 biți. Portul serial conține blocuri distincte pentru controlul transmisiei și recepției datelor de un octet. Registrele sale de transmisie și recepție sunt accesibile prin intermediul registrului SBUF din

SFR. O scriere în SBUF determină încărcarea registrului de transmisie, iar citirea SBUF accesează registrul de recepție.

(msb)				(lsb)				(98h)
SM0	SM1	SM2	REN	TB8	RB8	TI	RI	
SM0	SM1	Mod	Descriere	Viteza serială	TB8	RB8	TI	RI
0	0	0	registru cu deplasare	$f_{osc}/12$	- cel de-al 9-lea bit transmis în modurile 2 și 3.	- cel de-al 9-lea bit recepționat în modurile 2 și 3.	- fanion de întrerupere la transmisie. Este setat HW, în funcție de modul de operare. Trebuie resetat SW.	- fanion de întrerupere la recepție. Este setat HW, în funcție de modul de operare. Trebuie resetat SW.
0	1	1	UART - 8 biți	variabilă				
1	0	2	UART - 9 biți	$f_{osc}/32$ sau $f_{osc}/64$				
1	1	3	UART - 9 biți	variabilă				
SM2 - activează comunicația multiprocesor, în modurile 2 și 3.								
REN - activează recepția serială.								

Fig.6.23. SCON - registrul de control al portului serial

Modul 0 asigură o funcționare de tip registru de deplasare, având intrarea și ieșirea datelor pe linia RxD/P3.0, iar tactul de deplasare este primit/generat pe linia TxD/P3.1. Rata de comunicație în acest mod este fixă și are valoarea $f_{osc}/12$, fiind de 1Mbps la $f_{osc}=12$ MHz.

Transmisia este inițiată de orice instrucțiune care folosește ca registru destinație SBUF. De asemenea, blocul de control al transmisiei forțează în “1” logic cea de a 9-a poziție a registrului de deplasare.

Transmisia începe cu bitul cel mai puțin semnificativ al octetului de date și se face în ritmul de 1 bit/ciclu. Odată cu deplasarea biților la ieșire, în ritmul fixat prin TxD, registrul se completează cu zerouri. Când și cel mai semnificativ bit al cuvântului a fost transmis, cel de-al 9-lea bit, care a fost setat pe “1”, ajunge în poziția cea mai puțin semnificativă a registrului de deplasare și blocul de control dezactivează transmisia. Totodată se realizează o ultimă deplasare și se setează fanionul de întrerupere la transmisie - TI din SCON, în al 10-lea ciclu mașină.

Recepția este inițiată printr-o comandă cu REN=1 în SCON, precedată de resetarea bitului RI. Blocul de control al recepției înscrie biții 11111110b în registrul de deplasare. La fiecare bit recepționat pe linia RxD, în acest registru are loc o deplasare la stânga cu o poziție, tot în ritmul de 1 bit/ciclu stabilit prin TxD. Atunci când bitul “0” ajunge în poziția din extrema stângă a registrului de deplasare, blocul de control comandă încă o deplasare și încărcarea registrului SBUF cu valoarea recepționată. În cel de-al 10-lea ciclu mașină de la înscrierea comenzii în SCON recepția este oprită și se setează fanionul de întrerupere RI.

Modul 1 permite o funcționare de tip asincron (UART) cu un bit de start (“0” logic), 8 biți de date și un bit de stop (“1” logic). Viteza de comunicație este variabilă (între 110 Baud și 62,5 KBaud) și este controlată de Timerul 1 sau de Timerul 2 (numai la 8052). De obicei, Timerul 1 se folosește în modul 2, cu autoîncărcare (v. §6.3.4.1), caz în care viteza de comunicație se calculează cu relația [50] :

$$V_{[Baud]} = \frac{2^{SMOD}}{32} \times \frac{f_{osc}}{12[256 - (TH1)]},$$

în care SMOD este bitul cel mai semnificativ al registrului PCON (v.tab.6.5 și §6.3.7), iar (TH1) reprezintă valoarea de reîncărcare, memorată de TH1.

Pentru viteze mici de comunicație se preferă modul 1 de funcționare a Timer-ului 1, ca timer de 16 biți și reîncărcarea sa prin program, în rutina de tratare a întreruperii.

În tab.6.9 sunt prezentate vitezele de comunicație uzuale și modul de obținere a lor folosind Timerul 1. În cazul utilizării Timerului 2 (numai la 8052) ca generator de tact (v.tab.6.8), funcționarea acestuia este similară modului cu reîncărcare pe 16 biți, cu valoarea programată în registrele RCAP2H și RCAP2L. Deosebirea constă în faptul că de această dată incrementarea nu se realizează la fiecare ciclu mașină ($f_{osc}/12$), ci la fiecare stare mașină ($f_{osc}/2$).

În fig.6.24 este prezentată schema de funcționare a Timerului 2 ca generator de semnal de tact pentru Portul serial.

Viteza de comunicație [Baud]	f_{osc} (MHz)	Bitul SMOD	Timer 1		
			C/ \bar{T}	Mod	Valoarea în TH1
62,5 K	12	1	0	2	FFh
19,2 K	11,059	1	0	2	FDh
9,6 K	11,059	0	0	2	FDh
4,8 K	11,059	0	0	2	FAh
2,4 K	11,059	0	0	2	F4h
1,2 K	11,059	0	0	2	E8h
137,5	11,986	0	0	2	1Dh
110	6	0	0	2	72h
110	12	0	0	1	FEEDh

Tab.6.9 - Vitezele de comunicație prin folosirea Timerului 1

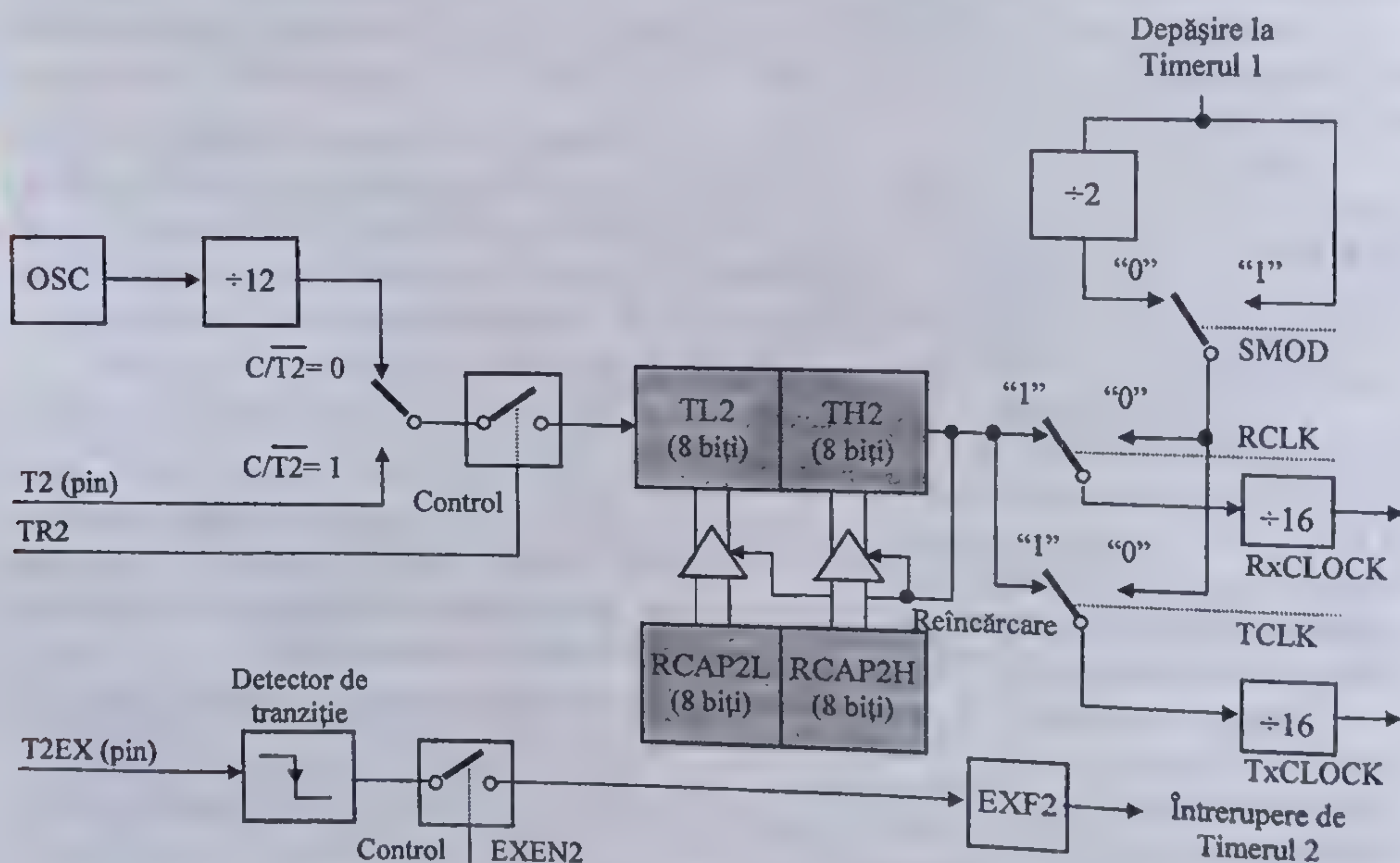


Fig.6.24. Timerul 2 - funcționarea ca generator de tact pentru Portul serial

Acest regim este activ numai dacă $RCLK+TCLK=1$; vitezele de comunicație pot fi simultan diferite la recepție și la transmisie. Și în acest caz viteza de comunicație este determinată de frecvența de apariție a depășirii în numărătorul TH2-TL2 și care, pentru

funcționarea de timer, se calculează cu formula [50]:

$$V_{[\text{Baud}]} = \frac{f_{\text{osc}}}{32 \times [65.536 - (\text{RCAP2H}, \text{RCAP2L})]}$$

unde (RCAP2H, RCAP2L) este conținutul registrelor cu același nume luat ca o constantă de 16 biți fără semn.

Transmisia în modul 1 este inițiată de o înscriere în registrul SBUF, care determină și o poziționare pe "1" logic al celui de-al 9-lea bit (TB8) din registrul de deplasare. Însă începerea efectivă a transmisiei este declanșată de următoarea depășire a unui contor divizor prin 16 (v.fig.6.24). După transmiterea datelor, în ritmul impus de TxCLOCK, în poziția cea mai semnificativă a registrului de deplasare ajunge cel de-al 9-lea bit. În această stare, blocul de control al transmisiei determină încă o deplasare și setează fanionul TI. Ca urmare, după cea de-a 10-a depășire din momentul înscrierii datei în SBUF, transmisia este dezactivată.

Recepția în modul 1 este activată de o tranziție 1→0 pe linia RxD dacă REN=1 în SCON. Pentru sesizarea tranziției, linia RxD este eșantionată de 16 ori într-o perioadă corespunzătoare vitezei de comunicație. Pentru fiecare bit de date recepționat, blocul de control comandă eșantionarea sistematică a liniei RxD de 3 ori succesiv, în cea de a 7-a, a 8-a și a 9-a stare din cele 16 ale număratorului de eșantionare. Este acceptată valoarea găsită pe linia RxD în ultimele 2 eșantionări din cele 3. Această procedură asigură rejecția perturbațiilor.

Dacă valoarea acceptată la eșantionarea primului bit pe linia RxD nu este "0" (bitul de start), logica de recepție se resetează și se așteaptă o nouă tranziție 1→0. Procedura menționată asigură rejecția unor biți de start falși. Dacă bitul de start este corect, acesta este transmis la intrarea registrului de deplasare și apoi a întregului cadru (data și bitul de stop). Când bitul de start ajunge în poziția extremă-stânga a registrului de deplasare, blocul de control al recepției determină încă o deplasare la stânga, încărcarea registrului SBUF cu octetul recepționat, setarea fanionului RI și încărcarea bitului de stop în RB8 din SCON. Încheierea recepției este realizată așa cum s-a arătat mai sus, dacă și numai dacă sunt îndeplinite condițiile:

1) RI = 0 și 2) SM2 = 0 sau STOP BIT = 1.

Dacă măcar una din aceste condiții nu este îndeplinită, cadrul recepționat se pierde și RI nu este setat. Indiferent dacă condițiile menționate sunt îndeplinite sau nu, unitatea de recepție trece din nou la testarea tranziției 1→0 pe linia RxD.

Modul 2 corespunde funcționării de tip UART cu 11 biți: un bit de start ("0"), 8 biți de date (lsb este primul), un al 9-lea bit de date programabil și un bit de stop ("1"). La transmisie, cel de-al 9-lea bit de date (TB8) poate fi forțat fie pe "0" fie pe "1". La recepție, cel de al 9-lea bit de date ajunge în RB8 din SCON.

Viteza de comunicație în modul 2 este selectabilă: $f_{\text{osc}}/32$ sau $f_{\text{osc}}/64$. Astfel, la $f_{\text{osc}}=12\text{MHz}$ viteza de comunicație maximă poate fi de 375 KBauds și cea minimă de 187,5 KBauds. În fig.6.25 este prezentată diagrama funcțională a Portului serial în acest mod.

Ca și în modul 1, **transmisia** este inițiată de orice instrucțiune care utilizează SBUF ca registru destinație. Un semnal "scrie în SBUF" realizează și încărcarea bitului TB8 din SCON în cea de-a 9-a poziție a registrului cu deplasare. Transmisia începe în S1P1 al următorului ciclu mașină determinat de depășirea unui contor divizor prin 16, fapt ce asigură sincronizarea transferului biților. Pe linia TxD se transmite bitul de start și apoi cei 8 biți de date, prin deplasare la dreapta. Când bitul TB8 este la ieșire, atunci bitul de stop ajunge chiar în poziția din registru a lui TB8. Celelalte poziții sunt completate cu zerouri. În această situație, blocul de control al transmisiei mai inițiază o ultimă deplasare, după care se dezactivează transmisia și se setează TI, în cea de-a 11-a stare a contorului divizor prin 16 de după semnalul "scrie în SBUF".

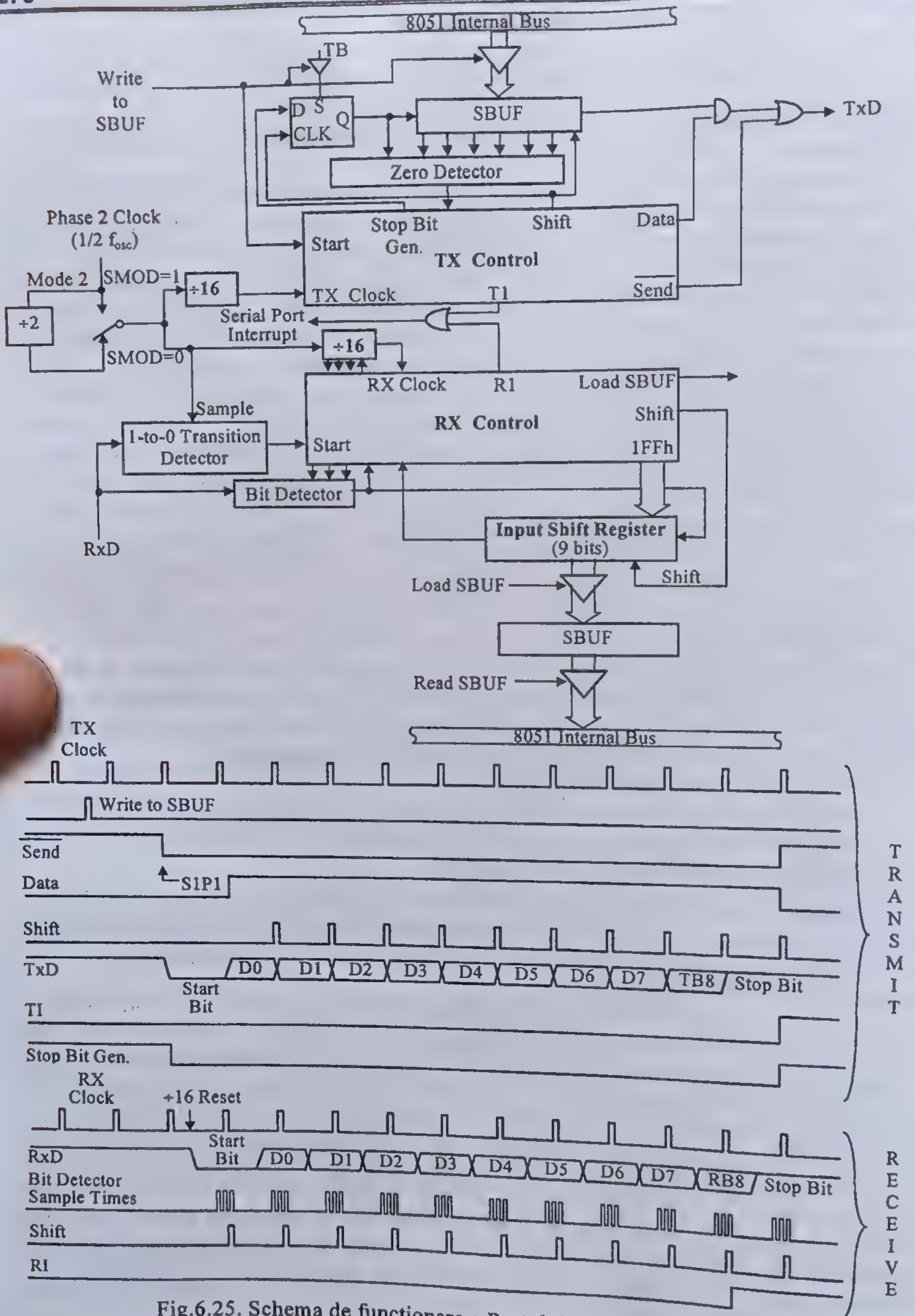


Fig.6.25. Schema de funcționare a Portului serial în modul 2

Recepția este inițiată la detectarea unei tranziții 1→0 pe linia RxD, dacă REN=1 în SCON. Ca și în modul 1, linia RxD este eșantionată cu o frecvență de 16 ori mai mare decât frecvența corespunzătoare vitezei de comunicație. La detectarea unei tranziții, contorul divizor prin 16 este resetat și în registrul de deplasare se înscrie 1Fh. Apoi începe recepția serie a celor 9 biți de date, de la dreapta la stânga, cu verificarea validității de 3 ori la fiecare bit (fig.6.25). Când bitul de start ajunge la extrema stângă a registrului de deplasare, acesta semnalizează blocului de control al recepției să execute o ultimă deplasare, încărcarea SBUF și RB8, și setarea RI. Semnalele de încărcare a SBUF și RB8 și de setare a fanionului RI vor fi generate numai dacă pe durata ultimei deplasări sunt îndeplinite simultan condițiile:

1) RI = 0 și 2) SM2 = 0 sau cel de-al 9-lea bit recepționat este "1".

Neîndeplinirea celor două condiții determină pierderea informației recepționate, iar RI nu este setat. Dacă ambele condiții sunt îndeplinite, cel de-al 9-lea bit recepționat este încărcat în RB8 din SCON, iar primii 8 biți de date sunt transferați în SBUF. După un tact, indiferent dacă condițiile mai sus menționate sunt sau nu îndeplinite, blocul de control al recepției reîncepe detectarea unei tranziții 1→0 pe linia RxD. De notat că valoarea recepționată a bitului de stop nu afectează în nici un fel SBUF, RB8 sau RI.

Modul 3 este identic cu modul 2, cu deosebirea că viteza de comunicație poate fi variabilă și stabilită de Timerul 1 sau de Timerul 2 (numai la seria 8052), la fel ca în modul 1.

6.3.4.3. Comunicația multiprocesor

Modurile 2 și 3 posedă facilități pentru realizarea comunicației seriale multipunct. După cum s-a văzut, în aceste moduri sunt recepționați 9 biți de date, cel de al 9-lea bit fiind transferat în RB8 din SCON.

Portul serial al unui microcontroler din familia MCS-51 poate fi programat astfel ca atunci când este recepționat un bit de stop, întreruperea portului să fie activată numai dacă RB8=1. Această facilitate se stabilește prin intermediul bitului SM2 din SCON. Pentru a ilustra funcționarea în regim multiprocesor se consideră schema din fig.6.26, în care procesorul MASTER comunică cu procesoarele SLAVE prin intermediul interfeței seriale. Deoarece liniile TxD și RxD au caracteristicile Portului 3, procesoarele SLAVE transmit către MASTER informația prin intermediul unei porți AND. Dacă liniile TxD ar avea facilitatea O.C. sau T.S., în locul porții AND ar fi suficient un rezistor conectat la VCC.

Când procesorul MASTER dorește să transmită un bloc de date unui procesor SLAVE, mai întâi transmite un octet de tip *adresă de identificare*. Un astfel de octet diferă de un octet de date prin faptul că cel de-al 9-lea bit este "1", pentru date fiind "0" logic. Dacă SM2=1, numai octeții de tip adresă activează întreruperea de recepție a Portului serial.

Inițial, toate procesoarele SLAVE lucrează cu SM2=1. Un octet de adresă transmis de MASTER va întrerupe toate procesoarele SLAVE, fiecare putând examina octetul primit pentru a verifica dacă îi este adresat. De exemplu, dacă MASTER-ul dorește să dialogheze cu SLAVE 3, atunci va transmite un cuvânt de forma 100000011B. SLAVE-ul adresat va recunoaște adresa (în exemplu, valoarea 03h) și își va reseta bitul SM2, pregătindu-se astfel pentru recepția datelor.

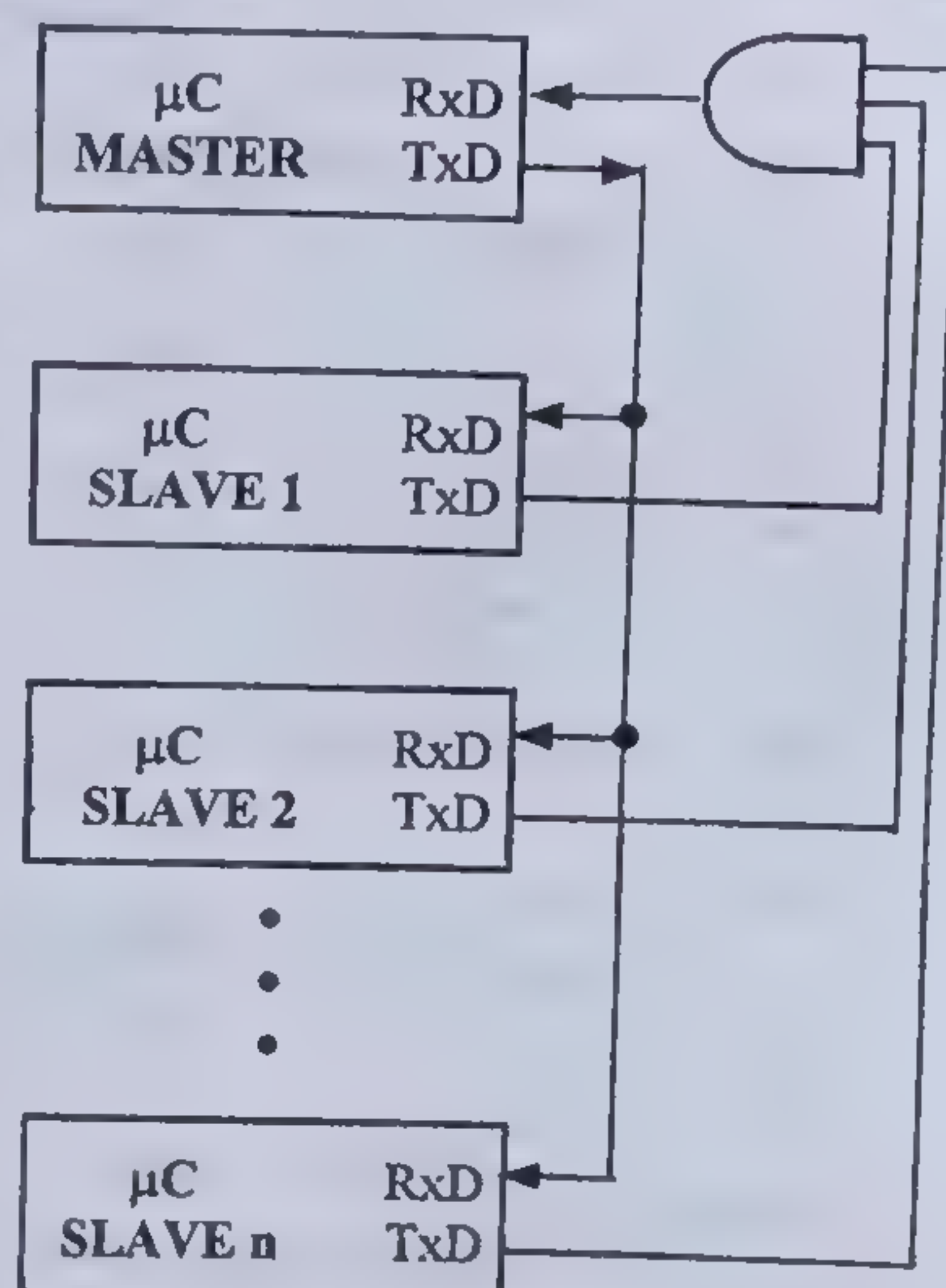


Fig.6.26. Schema comunicației multiprocesor

Celelalte procesoare SLAVE nu-și vor dezactiva bitul SM2 și vor rămâne în așteptarea adresei proprii, ignorând datele transmise de MASTER.

Bitul SM2 din SCON nu are nici un efect în modul 0, însă în modul 1 poate fi utilizat pentru verificarea validității bitului de stop. La recepție în modul 1, dacă $SM2 = 1$, întreruperea nu va fi activată decât dacă bitul de stop recepționat este valid ("1").

6.3.5. Sistemul de întreruperi

Familia MCS-51 posedă o logică "on chip" de control pentru întreruperi multiple generate de resursele interne sau din exterior. Varianta 8051 are 5 surse, iar 8052 are 6 surse de întreruperi, toate fiind mascabile. În fig.6.27 sunt prezentate sursele de întreruperi și modul în care acestea sunt controlate de către CPU. Celor 5(6) surse le corespund locațiile de tratare și prioritățile implicite din tab.6.10.

Sursa de întreruperi	Adresa de tratare	Prioritatea
IE0 - de la $\overline{INT0}$	0003h	<div style="text-align: center;"> maximă ↓ minimă </div>
TF0 - depășire Timer 0	000Bh	
IE1 - de la $\overline{INT1}$	0013h	
TF1 - depășire Timer 1	001Bh	
RI+TI - Portul serial	0023h	
TF2+EXF2 - Timer 2 (numai la seria 8052)	002Bh	

Tab.6.10. Surse de întreruperi la MCS-51

Întreruperile sunt vectorizate și vectorilor respectivi le sunt rezervate locații fixe de tratare aflate la intervale de 8 octeți, începând cu adresa 0003h.

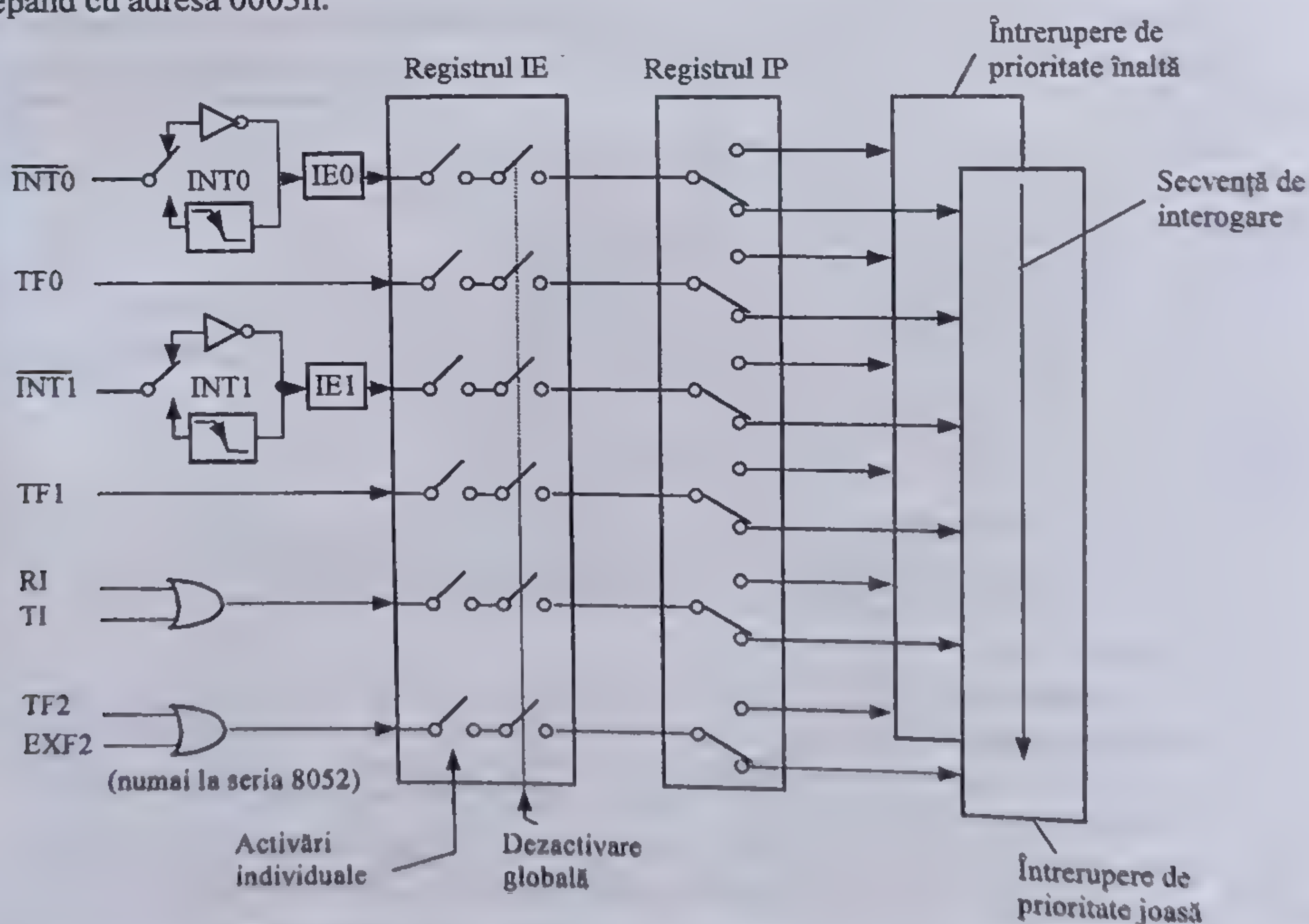


Fig.6.27. Sistemul de control al întreruperilor la MCS-51

Întreruperilor externe $\overline{INT0}$ și $\overline{INT1}$ li se poate programa modul de prezentare a cererii (activare pe nivel sau pe front) prin intermediul biților IT0 și respectiv IT1 din TCON (v.fig.6.17). Prin intermediul registrului IE (Interrupt Enable), adresabil la nivel de bit (fig.6.28), pot fi activate/inhibate individual sau global toate cele 5(6) surse de întreruperi.

(msb)				(lsb)				
EA	-	ET2	ES	ET1	EX1	ET0	EX0	(A8h)
Simbol	Poziția	Funcția						
EA	IE.7	Validează /inhibă toate întreruperile. Dacă $\overline{EA} = 0$, nu va fi acceptată nici o cerere. Dacă $\overline{EA} = 1$, fiecare sursă de întreruperi va fi validată/inhibată individual, prin biții proprii.						
-	IE.6	Rezervat						
ET2	IE.5	Bitul de validare a întreruperii Timerului 2 (numai la seria 8052).						
ES	IE.4	Bitul de validare a întreruperii Portului serial.						
ET1	IE.3	Bitul de validare a întreruperii Timerului 1						
EX1	IE.2	Bitul de validare a întreruperii externe $\overline{INT1}$						
ET0	IE.1	Bitul de validare a întreruperii Timerului 0.						
EX0	IE.0	Bitul de validare a întreruperii externe $\overline{INT0}$.						

Fig.6.28. IE - registrul pentru validarea întreruperilor

Trebuie menționat faptul că biții neutilizați IE.6 și IE.5 (ultimul numai la seria 8051) nu pot fi înscrși cu "1" logic la programarea registrului IE.

Sistemul de întreruperi este prevăzut și cu posibilitatea programării individuale a fiecărei surse de întreruperi pe unul din cele două niveluri de prioritate permise. Acest lucru este posibil prin intermediul registrului IP (Interrupt Priority), prezentat în fig.6.29. Și acest registru, având adresa B8h, este adresabil la nivel de bit; biții neutilizați trebuie să nu fie poziționați pe "1" logic.

(msb)				(lsb)				
-	-	PT2	PS	PT1	PX1	PT0	PX0	(B8h)
Simbol	Poziția	Funcția						
-	IP.7	Rezervat						
-	IP.6	Rezervat						
PT2	IP.5	Bitul de prioritate a întreruperii Timerului 2 (numai la seria 8052).						
PS	IP.4	Bitul de prioritate a întreruperii Portului serial.						
PT1	IP.3	Bitul de prioritate a întreruperii Timerului 1						
PX1	IP.2	Bitul de prioritate a întreruperii externe $\overline{INT1}$						
PT0	IP.1	Bitul de prioritate a întreruperii Timerului 0.						
PX0	IP.0	Bitul de prioritate a întreruperii externe $\overline{INT0}$.						

Fig.6.29. IP - registrul de prioritate a întreruperilor

În cazul în care apar simultan două cereri de întrerupere aflate pe niveluri diferite de prioritate, va fi servită cererea de prioritate ridicată. Atunci când apar simultan cereri de pe același nivel de prioritate, se declanșează o secvență internă de interogare ("polling sequence") care determină cererea ce va fi servită. Acest procedeu asigură în interiorul fiecărui nivel de prioritate rezolvarea ordinii de servire, în conformitate cu specificațiile din tab.6.10.

Existența cererilor de întrerupere este testată în fiecare ciclu mașină, în starea S2P2, starea lor fiind memorată în sistemul de control al întreruperilor. În ciclul următor se rezolvă prioritățile (excepție face TF2, care este setat în S2P2, fiind interogată în același ciclu) și se forțează hardware execuția unei instrucțiuni LCALL *addr* (Long CALL). Această instrucțiune salvează registrul PC în stivă (dar nu și PSW-ul) și îl încarcă apoi cu adresa *addr*, corespunzătoare sursei de întrerupere, specificată în tab.6.10. În funcție de sursa care a generat întreruperea, odată cu salvarea registrului PC se resetează automat (HW) și fanionul de întrerupere asociat. Astfel, fanioanele IE0 și IE1 (fig.6.27) sunt resetate numai dacă întreruperile externe sunt active pe front. În schimb, fanioanele de întrerupere ale Portului serial și Timerului 2 nu sunt niciodată resetate automat. Acestea trebuie resetate explicit prin program, în rutinele de servire a întreruperilor, după ce este identificată cu exactitate cauza care a produs întreruperea. Revenirea din întrerupere trebuie realizată cu o instrucțiune RETI, care permite sistemului de control al întreruperilor să accepte cereri aflate în așteptare, de aceeași prioritate sau mai mică (v. §6.2).

Timpul de răspuns la o întrerupere poate fi de minimum 3 cicluri și maximum 9 cicluri. Valoarea maximă este atinsă atunci când întreruperea apare în timpul execuției celor mai lungi instrucțiuni (MUL sau DIV), sau atunci când o instrucțiune RETI este în curs de execuție ori are loc un acces la registrele IE sau IP. În unele aplicații sunt necesare mai mult de două niveluri de prioritate - cât asigură hardware familia MCS-51 clasică. În astfel de cazuri poate fi realizat software un al treilea nivel de prioritate, printr-un program simplu, care consumă numai $10\mu s$ la $f_{osc} = 12MHz$.

```

PUSH IE
MOV IE,#MASK
ACALL etich
..... ; corpul rutinei
POP IE
RET
etich: RETI

```

Mai întâi, întreruperile de prioritate superioară lui 1 (pe care le vom numi de prioritate 2) sunt asignate în IP ca întreruperi de prioritate 1. Rutinele de servire pentru întreruperile de prioritate 1, care se consideră a fi întreruptibile de întreruperile de "prioritate 2", au inclus secvența de instrucțiuni de mai sus. Imediat ce o întrerupere de prioritate 1 este acceptată, registrul IE se redefineste astfel încât să dezactiveze toate întreruperile în afară de cele de "prioritate 2". Apoi, printr-un ACALL la adresa *etich* se execută RETI, care determină resetarea bistabilului întreruperii de prioritate 1 în curs de servire. În acest punct al programului orice întrerupere de prioritate 1 care este activă poate fi servită, dar active sunt numai întreruperile care au rămas validate, adică cele "de prioritate 2". Revenirea din rutină trebuie să se facă cu o instrucțiune RET, normală, nu printr-o nouă instrucțiune RETI.

Necesitatea reală de creștere a numărului de niveluri de prioritate în tratarea întreruperilor a făcut ca la cele mai recente produse din această categorie (8XC51FX, 8XC52/54/58 și 8XC51GB) să fie introdus un al doilea registru de priorități, IPH, fapt ce permite creșterea numărului nivelurilor de prioritate la 4.

6.3.5.1. Funcționarea pas cu pas prin întreruperi

Specificul structurii sistemului de întreruperi al familiei MCS-51 permite realizarea unei execuții pas cu pas a unui program, prin utilizarea unui soft de sistem minimal. Din cele arătate până acum s-a văzut că tratarea unei întreruperi nu poate fi abandonată în favoarea unei cereri de

aceeași prioritate. În plus, la revenirea dintr-o rutină de tratare, prin RETI, nu se acceptă nici o altă cerere decât după execuția următoarei instrucțiuni din program.

O modalitate simplă de a folosi cele arătate mai sus pentru funcționarea pas cu pas, constă în a programa una din întreruperile externe (de exemplu $\overline{\text{INT0}}$) să fie activă pe nivel. Rutina corespunzătoare de servire a întreruperii se va încheia cu următoarea secvență:

JNB	P3.2,\$; Așteaptă aici până când $\overline{\text{INT0}}$ trece în "1"
JB	P3.2,\$; Acum așteaptă aici până când $\overline{\text{INT0}}$ trece în "0"
RETI		; Revine și mai execută o instrucțiune din programul întrerupt.

Dacă pinul P3.2, care este linia $\overline{\text{INT0}}$, va fi menținut la "0" logic, atunci CPU va intra în rutina de întrerupere aferentă până când $\overline{\text{INT0}}$ va pulsa la "1" și apoi la "0". Atunci se va executa RETI, cu revenire în programul întrerupt și va executa o instrucțiune. Deoarece pinul P3.2 a revenit la "0" se va intra din nou în rutina de tratare a întreruperii externe 0, așteptându-se un nou impuls pe linia P3.2 ș.a.m.d. Rezultă astfel că la fiecare impuls complet pe linia $\overline{\text{INT0}}$ va fi executată o singură instrucțiune (un pas) din programul principal.

6.3.6. Resetarea microcontrolerelor familiei MCS-51

Resetarea se obține prin forțarea liniei RST (v.fig.6.2) la "1" pe durata a două cicluri mașină (24 de perioade ale oscilatorului intern), în timp ce oscilatorul este în funcțiune. Unitatea centrală de procesare răspunde prin generarea unui reset intern, așa cum se arată în fig.6.30. În fiecare ciclu mașină, linia RST este eșantionată pe durata fazei 2 din starea a 5-a (S5P2). Pinii

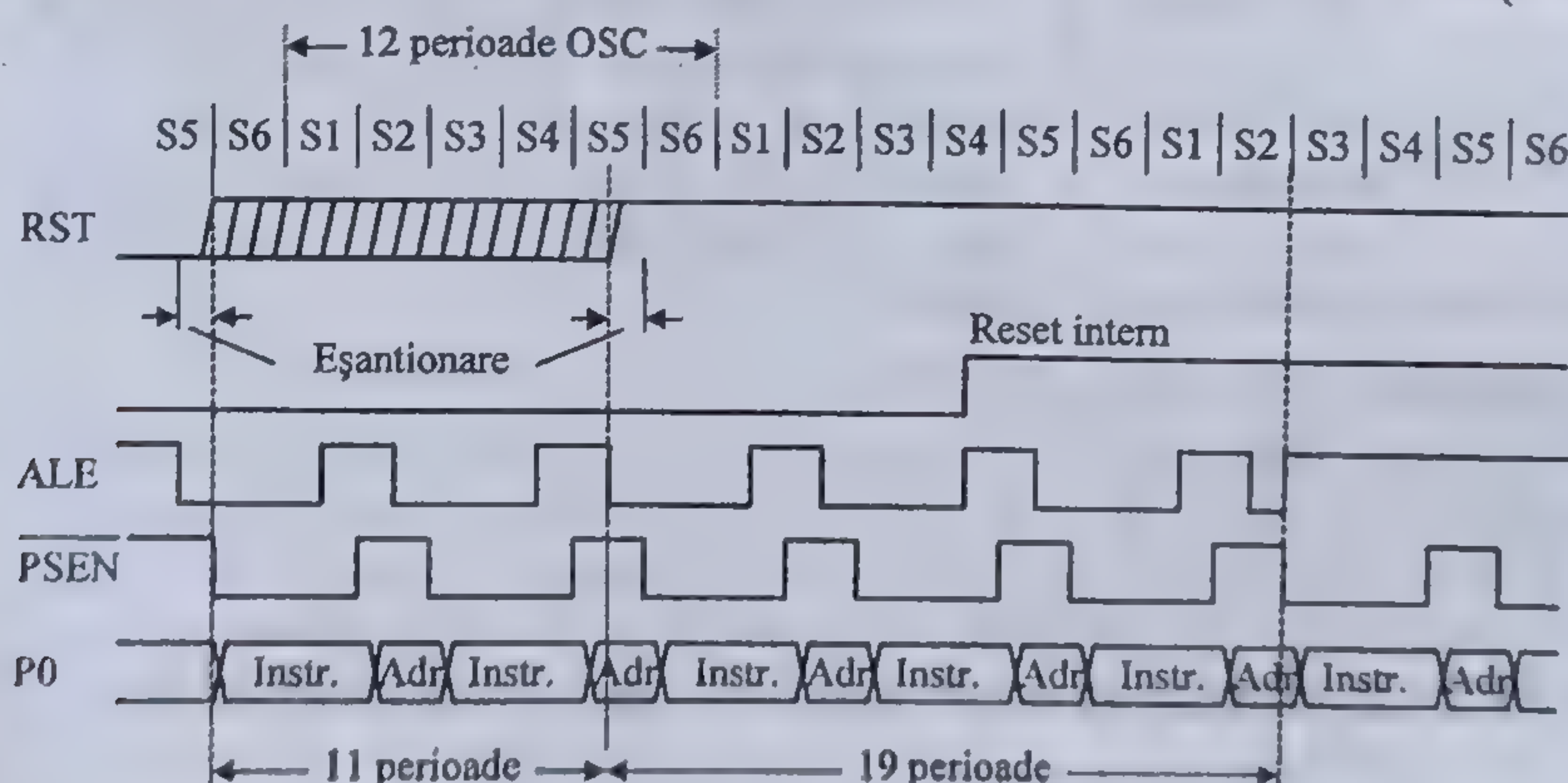


Fig.6.30. Diagrama de semnale la resetare

porturilor vor păstra valoarea corespunzătoare activității curente încă 19 perioade de oscilator după ce un "1" logic este detectat la pinul RST. De asemenea și semnalele ALE și $\overline{\text{PSEN}}$ sunt trecute pe "1" logic, pentru a se evita intrarea microcontrolerului într-o stare nedeterminată. Conținutul memoriei RAM interne nu este afectat de resetare, ci numai de dispariția alimentării.

Semnalul intern de resetare determină forțarea tuturor registrelor din zona SFR în "0" logic, cu excepția registrelor SFR ale porturilor, a registrului SP și a registrului SBUF. Latch-urile porturilor sunt inițializate cu FFh, registrul SP cu 07h, iar conținutul lui SBUF este nedeterminat. De asemenea, unii biți ai registrelor IP, IE și PCON rămân nedefiniți.

În tab.6.11 sunt sintetizate valorile care se află în registrele din zona SFR imediat după resetare.

Numele SFR	Valoarea după resetare
PC	0000h
ACC	00h
B	00h
PSW	00h
SP	07h
DPTR	0000h
P0-P3	FFh
IP (8051)	xxx00000b
IP (8052)	xx000000b
IE (8051)	0xx00000b
IE (8052)	0x000000b
TMOD	00h
TCON	00h
TH0	00h
TL0	00h
TH1	00h
TL1	00h
TH2 (8052)	00h
TL2 (8052)	00h
RCAP2H (8052)	00h
RCAP2L (8052)	00h
SCON	00h
SBUF	Nedeterminată
PCON (HMOS)	0xxxxxxx b
PCON (CMOS)	0xxx0000b

Tab.6.11. SFR după resetare

Resetarea la conectarea alimentării

Resetarea microcontrolerelor familiei la aplicarea tensiunii de alimentare se realizează prin conectarea la intrarea RST a unor elemente pasive, care să asigure timpii necesari inițializării corecte.

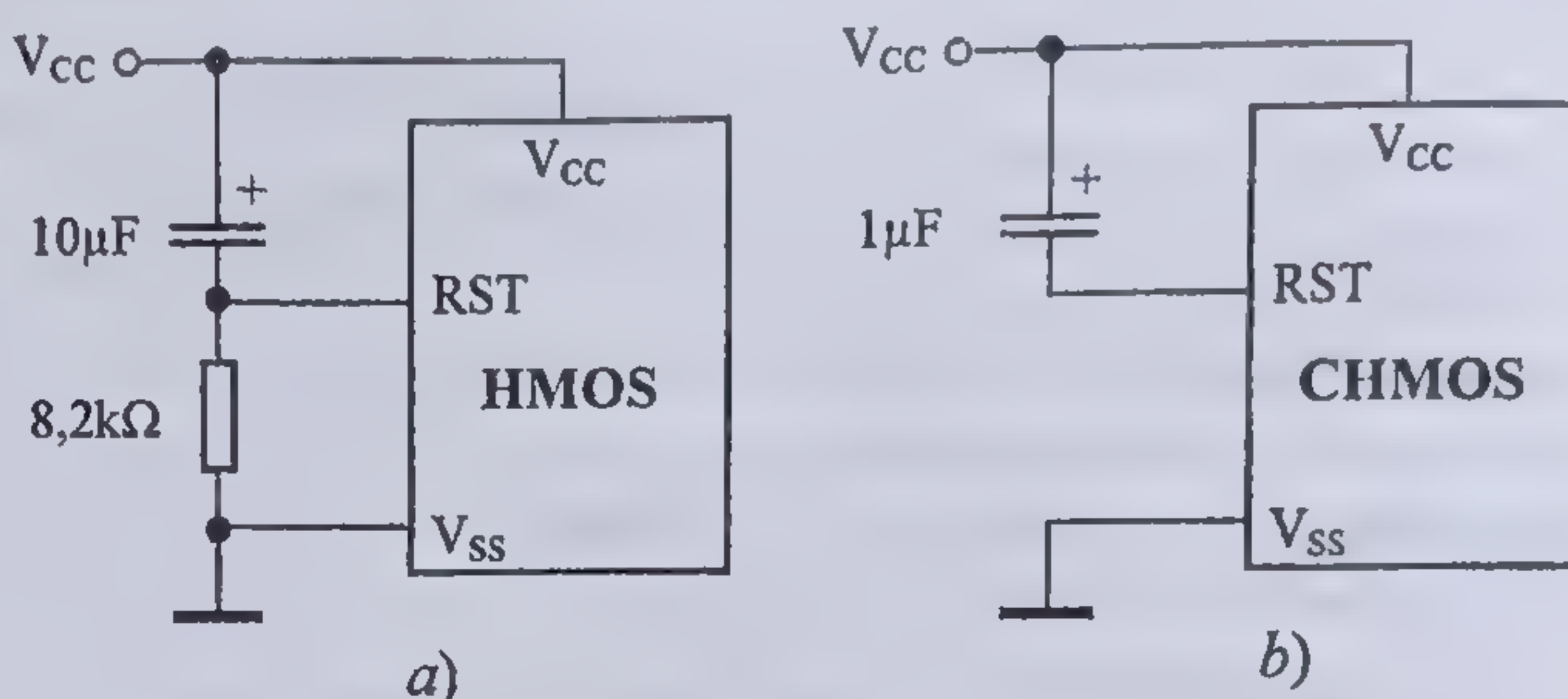


Fig.6.31. Resetarea MCS-51 la conectarea alimentării

Componența acestor circuite depinde de tipul de tehnologie în care este realizat microcontrolerul. Astfel, pentru dispozitivele în tehnologie HMOS se aplică schema din fig.6.31a, iar pentru cele în tehnologie CHMOS schema din fig.6.31b, întrucât acestea posedă un rezistor intern corespunzător de "pull-down". Circuitele de resetare asigură menținerea la V_{CC} a liniei RST un timp suficient care să asigure pornirea oscilatorului intern și apoi încă două cicluri mașină. Apariția tensiunii de alimentare fără o resetare corectă poate determina CPU să înceapă execuția instrucțiunilor de la o locație nedeterminată. Aceasta deoarece zona SFR, în speță registrul PC, poate să fie incorect inițializată.

6.3.7. Regimuri de consum redus

În aplicațiile care impun un consum redus de putere, pentru versiunea CHMOS de microcontrolere sunt prevăzute două regimuri de funcționare, cu reducerea puterii consumate: regimul de inactivitate ("idle mode") și regimul deconectat ("power down mode"). Versiunea HMOS nu are implementată facilitatea de consum redus. Controlul regimurilor de consum redus se realizează prin intermediul registrului PCON, a cărei configurație este prezentată în fig.6.32.

(msb)				(lsb)			
SMOD	-	-	-	GF1	GF0	PD	IDL

(87h)

Simbol	Poziția	Denumire și funcție
SMOD	PCON.7	Bit stabilire viteză comunicație Port serial în modurile 1, 2 sau 3.
-	PCON.6	Rezervat
-	PCON.5	Rezervat
-	PCON.4	Rezervat
GF1	PCON.3	Fanion de uz general
GF0	PCON.2	Fanion de uz general
PD	PCON.1	Bit stabilire mod "deconectat" (Power Down bit)
IDL	PCON.0	Bit stabilire mod "inactiv" (Idle mode bit)

Fig.6.32. PCON - registru de control a puterii consumate

Despre bitul SMOD s-a discutat în §6.3.4.2, la stabilirea vitezei de comunicație pentru Portul serial, atunci când Timerul 1 este utilizat pentru generarea semnalului de tact. La versiunile HMOS în PCON există numai acest bit.

Următorii trei biți semnificativi ai PCON nu sunt folosiți și nu trebuie niciodată înscrisi cu "1" logic. Acești biți sunt rezervați pentru variantele îmbunătățite ale familiei MCS-51.

Biții PCON.3 și PCON.2 sunt utilizați ca fanioane de uz general ("general-purpose flag bits"), fie în legătură cu regimul de consum redus, fie în alte scopuri.

Ultimii doi biți mai puțin semnificativi permit, la setare, definirea modului de funcționare cu consum redus: PD și IDL. Acești biți controlează funcționarea microcontrolerelor printr-o schemă de tipul celei prezentate în fig.6.33. Dacă în PCON sunt setați atât bitul PD, cât și bitul IDL, prioritară este comanda PD.

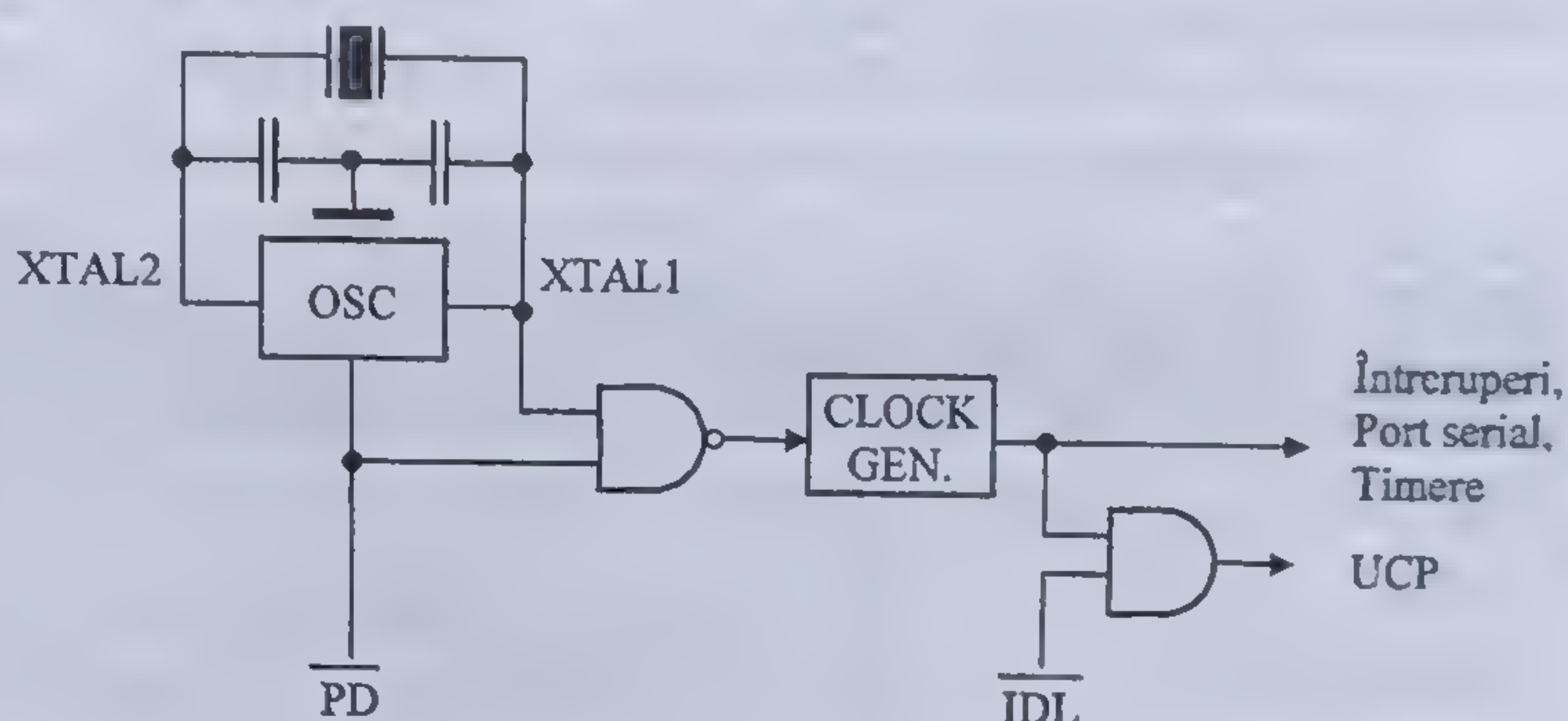


Fig.6.33. Schema de control a modurilor de consum redus

6.3.7.1. Modul inactiv (Idle mode)

Modul inactiv se obține prin setarea în PCON a bitului PCON.0. În acest caz se blochează semnalul intern de tact spre CPU, dar el se transmite către sistemul de întreruperi, portul serial și timere (fig.6.33). Ca urmare, pe durata modului inactiv starea CPU este blocată, cu menținerea datelor din registre. Semnalele la porturi sunt păstrate la valorile lor logice, iar semnalele ALE și $\overline{\text{PSEN}}$ se mențin la nivel ridicat.

Există două posibilități de ieșire din modul Idle: activarea unei întreruperi validate, care automat va șterge bitul PCON.0, respectiv prin reset hardware, care va aduce registrul PCON în starea inițială (v.tab.6.11). Fanioanele GF0 și GF1 pot fi utilizate pentru a indica dacă o întrerupere a apărut la funcționarea normală sau pe durata regimului Idle.

6.3.7.2. Modul deconectat (Power Down mode)

Modul deconectat se stabilește printr-o instrucțiune care determină setarea bitului PCON.1 în PCON. În acest mod, oscilatorul de tact de pe cip este oprit (fig.6.33), ceea ce determină stoparea completă a funcționării microcontrolerului. Informația din zona SFR și din întreg RAM-ul intern nu sunt alterate; de asemenea, nici valorile de la ieșirile porturilor. Semnalele ALE și $\overline{\text{PSEN}}$ sunt aduse, în acest caz, la nivel coborât.

Ieșirea din modul PD se face numai prin resetarea cipului, operație care reinițializează zona SFR, dar nu afectează RAM-ul intern.

6.3.8. Programarea, verificarea și protecția memoriei EPROM

La microcontrolerele cu memorie EPROM pe cip, pentru programare este necesar să fie respectate anumite condiții, specifice fiecărui tip de dispozitiv. În tab.6.12 sunt prezentate principalele caracteristici necesare programării memoriei EPROM interne [50], [51].

Normal, \overline{EA}/V_{PP} se menține la nivel logic "1" până înaintea activării semnalului ALE/PROG la "0" pentru 50 ms. Atunci \overline{EA}/V_{PP} se aduce la +21V, iar după impulsul ALE/PROG, revine la "1" logic.

Dacă bitul de securizare nu a fost programat, conținutul EPROM-ului poate fi verificat, atât în timpul cât și după programare. Condițiile necesare verificării sunt prezentate în tab.6.13 și în fig.6.34b (Portul 0 prevăzut cu rezistoare la V_{CC}), iar succesiunea semnalelor este dată în fig.6.35. Practic, schema de verificare este identică cu cea de programare, cu excepția liniei P2.7 care se aduce la "0" logic sau este folosită pentru un semnal de citire (ENABLE în fig.6.34b).

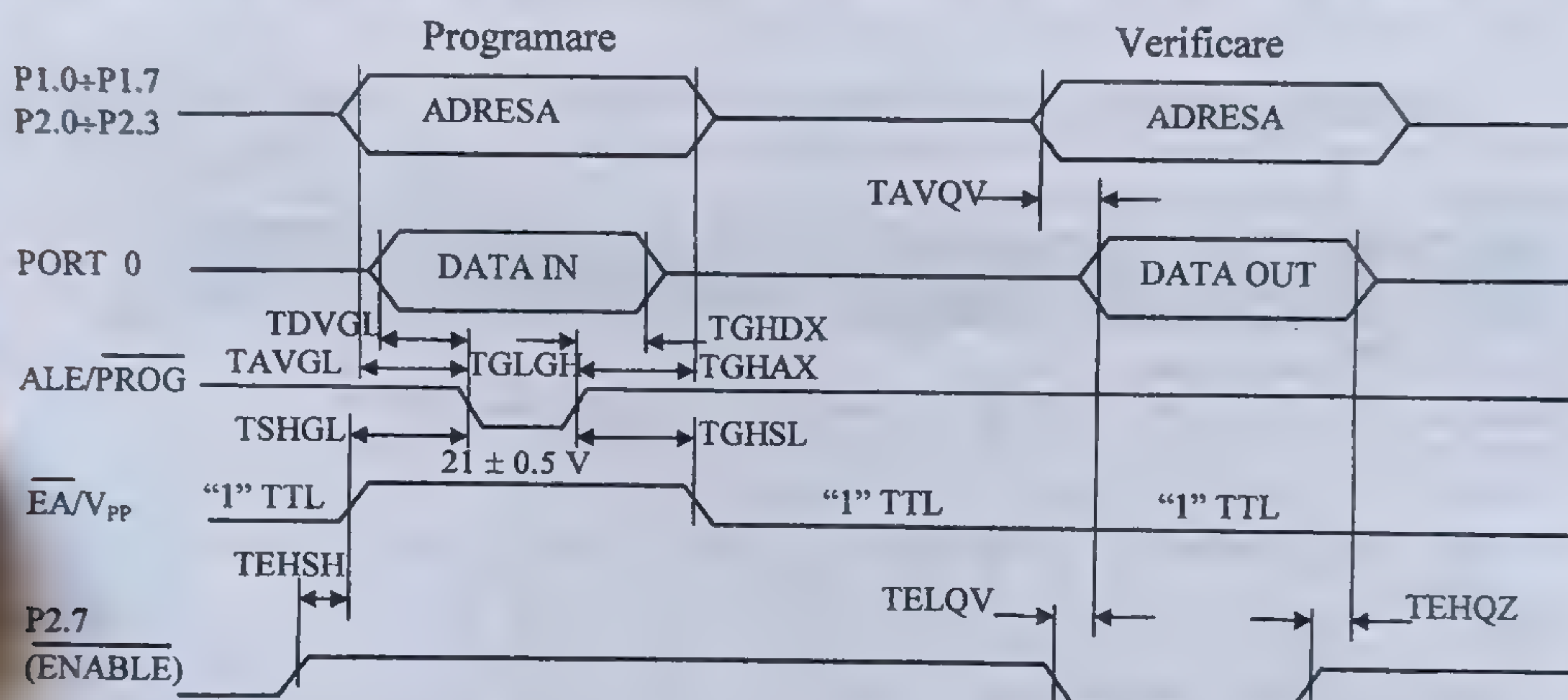


Fig.6.35. Secvențele semnalelor la programarea și verificarea memoriilor EPROM

Caracteristicile tehnice corespunzătoare figurii 6.35 sunt date în tab.6.14, pentru $V_{CC}=5V \pm 10\%$; $V_{SS}=0V$ și $T_A=21 \div 27^\circ C$ [51]. Mai trebuie specificat faptul că sursa de alimentare a

Simbol	Parametrul	Min.	Max.	Unitatea
V_{PP}	Tensiunea sursei de programare	20,5	21,5	V
I_{PP}	Curentul sursei de programare		30	mA
$1/TCLCL$	Frecvența oscilatorului	4	6	MHz
TAGVL	Setup-ul adresei la $\overline{PROG}=0$	48TCLCL		
TGHAX	Menținere adresă după \overline{PROG}	48TCLCL		
TDVGL	Setup-ul datei la $\overline{PROG}=0$	48TCLCL		
TGHDX	Menținere dată după \overline{PROG}	48TCLCL		
TEHSH	P2.7(\overline{ENABLE})=1 la V_{PP}	48TCLCL		
TSHGL	Setup-ul V_{PP} la $\overline{PROG}=0$	10		μs
TGHSL	Menținere V_{PP} după \overline{PROG}	10		μs
TGLGH	Durata \overline{PROG}	45	55	ms
TAVQV	Adresa-data validă		48TCLCL	
TELQV	$\overline{ENABLE}=0$ la data validă		48TCLCL	
TEHQZ	Data flotează după \overline{ENABLE}	0	48TCLCL	

Tab.6.14. Caracteristicile specifice programării și verificării EPROM la 8751H

programatorului trebuie să fie bine reglată și să nu permită generarea de vârfuri accidentale, peste valoarea de 21,5V, deoarece acestea pot duce la deteriorarea microcontrolerului.

Pentru *protecția* conținutului memoriei EPROM se folosește un bit de securizare (Lock Bit), care poate fi programat în conformitate cu specificațiile din tab.6.13 și fig.6.34c. Se poate observa că distincția față de modul "programare" se realizează prin linia P2.6, care la securizare trebuie menținută la "1" logic. După programarea bitului de protecție este blocat orice acces din exterior la memoria program "on chip": citire sau programare. De asemenea, nu mai poate fi executat nici un program în exteriorul microcontrolerului. Bitul de protecție o dată programat, nu mai poate fi resetat, decât prin ștergerea conținutului memoriei cu radiații ultraviolete. După ștergere, toți biții memoriei EPROM devin "1" logic.

Versiunile 8751BH și 8752BH necesită, de asemenea, pentru programarea memoriei EPROM interne o frecvență a oscilatorului de tact cuprinsă între 4 și 6 MHz. Modurile de programare și schema aferentă sunt prezentate în tab.6.15 și în fig.6.36 [51].

Modul	RST	PSEN	ALE/ PROG	EA/ V _{PP}	P2.6	P2.7	P3.3	P3.6	P3.7
Programare date	1	0	0*	V _{PP}	0	1	1	1	1
Verificare date	1	0	1	1	0	0	0	1	1
Programare criptare, adrese: 0÷1F (3F)*	1	0	0*	V _{PP}	0	1	1	0	1
Programare x=1	1	0	0*	V _{PP}	1	1	1	1	1
biți pro- x=2	1	0	0*	V _{PP}	1	1	1	0	0
tecție (LBx) x=3	1	0	0*	V _{PP}	1	0	1	1	0
Citire semnătură	1	0	1	1	0	0	0	0	0

Notă:
V_{PP} = 12,75 ± 0,25 V
*ALE este pulsă în "0" pentru 100 μs, de 25 ori "1" și "0"-nivele logice TTL

Tab.6.15. Modurile de programare a memoriei EPROM pentru 875xBH și 87C51(*)

În modul *programare*, pe linia $\overline{\text{PROG}}$ este necesar să se aplice câte 25 de impulsuri de 100 μs pentru fiecare octet, iar $\overline{\text{EA}}$ trebuie să fie menținută la 12,75V.

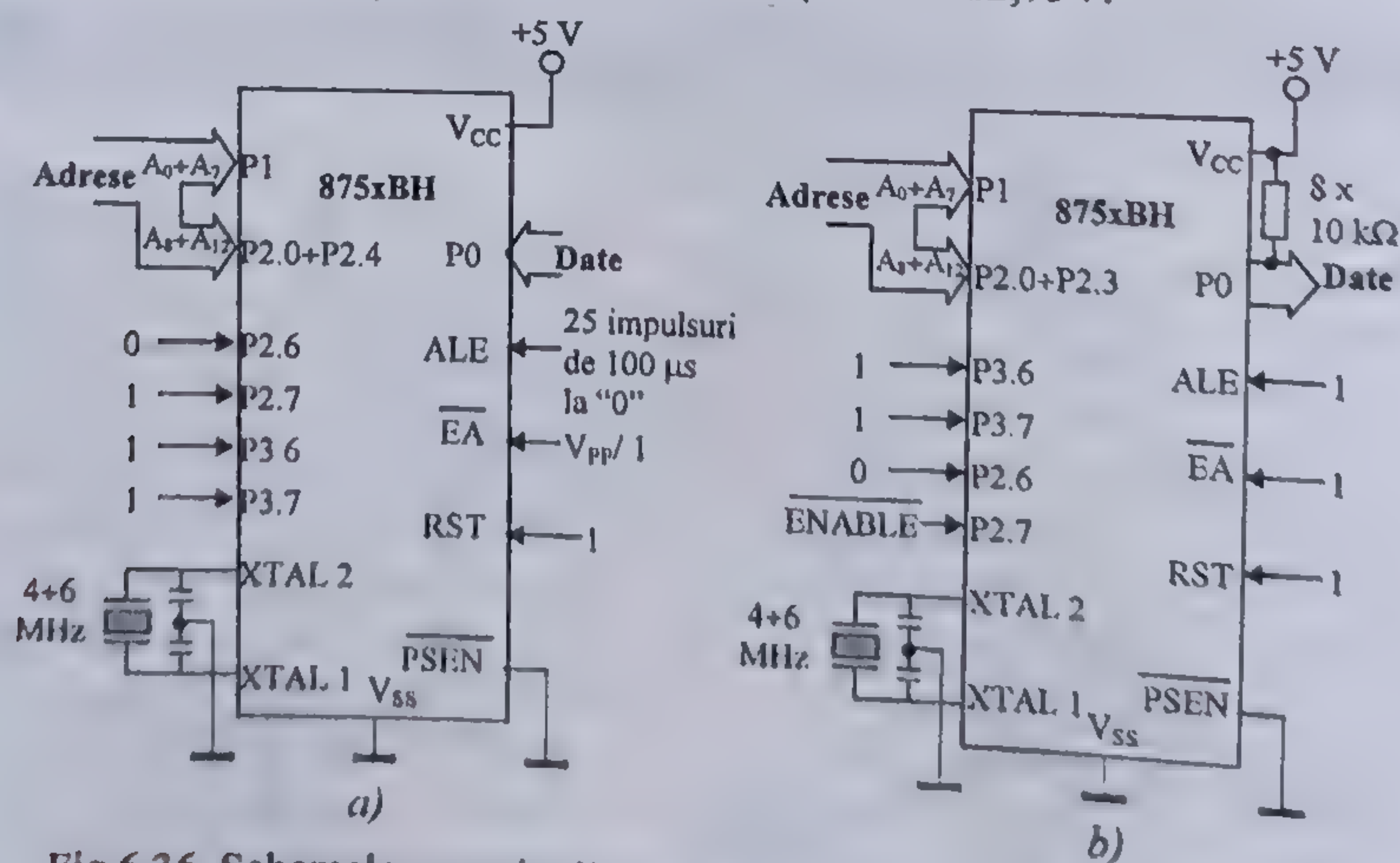


Fig.6.36. Schemele conexiunilor la 875xBH pentru: a) programare cod; b) verificare date

Secvența de programare este similară celei de la 8751H (v. fig.6.35), cu diferența impusă semnalului ALE/PROG, care trebuie să aibă succesiunea din fig.6.37. Acest mod de programare, denumit de producător "Quick-Pulse Programming Algorithm" [50], [51], permite o reducere considerabilă a duratei de înscriere a EPROM-ului la 13 s pentru 8751BH (4 Ko) și 26 s pentru 8752BH (8 Ko) (v. și tab.6.12).

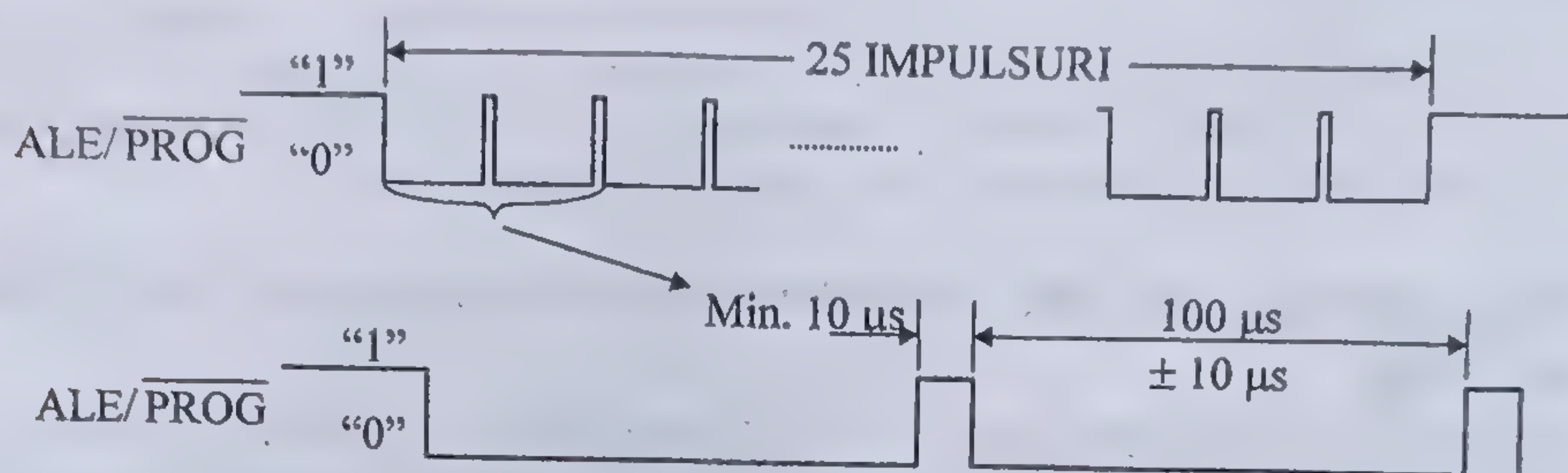


Fig.6.37. Forma de undă pentru semnalul PROG

Verificarea datelor programate în EPROM poate fi realizată atât în timpul programării cât și după programare, numai dacă nu au fost programați biții de securizare. Condițiile ce trebuie îndeplinite în modul verificare sunt prezentate în tab.6.15 și în fig.6.36b.

Spre deosebire de versiunea 8751H, la 875xBH există două modalități de securizare a conținutului memoriei EPROM: a) zonă de criptare și b) biții de protecție. Pentru zona de criptare, în EPROM sunt prevăzuți 32 octeți, între adresele 00h și 1Fh, care se programează în conformitate cu tab.6.15. Octetul citit la verificare reprezintă rezultatul funcției logice XNOR (eXclusive-NOR) între un octet din zona de criptare (adresat cu ultimii 5 biți ai adresei) și codul de date programat. Din acest motiv, la verificarea conținutului memoriei este necesar ca utilizatorul să cunoască codul de criptare pentru a putea decodifica corect datele citite. Tipurile 875xBH posedă 2 biți de protecție (Lock Bits) ale căror combinații realizează diferite grade de securizare, în conformitate cu tab.6.16a. Ștergerea memoriei conduce la pierderea securizării; atât zona de criptare cât și biții de protecție se șterg.

Biți de protecție		Logica activată	Biți de protecție				Logica activată
LB1	LB2			LB1	LB2	LB3	
N	N	Protecție minimă (citirea datelor este protejată numai prin zona de criptare)	1	N	N	N	Protecție minimă (verificarea datelor protejată numai prin zona de criptare)
P	N	Dezactivează programarea EPROM; instrucțiunile MOVC nu se execută; \overline{EA} este eșantionat și memorat la resetare	2	P	N	N	Dezactivează programarea EPROM; instrucțiunile MOVC nu se execută; \overline{EA} este eșantionat și memorat la resetare
P	P	Idem, dar se dezactivează și modul "Verificare"	3	P	P	N	Idem ca 2, dar se dezactivează și modul "Verificare"
N	P	Rezervat	4	P	P	P	Idem ca 3, dar se dezactivează și execuția din exterior

a)

P - programat; N - neprogramat

b)

Tab.6.16. Biții de protecție și efectul lor: a) la 875xBH și b) la 87C51

Memoriile EPROM ale microcontrolerelor 875xBH sunt prevăzute cu octeți de semnătură (signature bytes), pentru identificare. La citirea locațiilor 030h și 031h se returnează: (030h)=89h - indică produs Intel; (031h)=51h - indică tipul 8751BH, 52h - 8752BH.

Versiunea 87C51 [54] se programează similar cu tipurile 875xBH, deosebirea constând în faptul că sunt necesare numai 5 impulsuri de 100 μ s pentru un octet și se mai utilizează și linia P3.3 (v.tab.6.15). La verificare procedura este identică, însă pentru securizare se utilizează o zonă de criptare de 64 octeți și 3 biți pentru protecție (LB1, LB2, LB3) - tab.6.16b. Restul combinațiilor celor 3 biți nu sunt definite. Acest tip de microcontroler are 3 octeți de semnătură în locațiile 30h, 31h și 60h, returnând la citire valorile: 89h, 58h și 51h.

6.4. Facilitățile noi ale versiunilor perfecționate ale familiei MCS-51

Așa cum s-a menționat în §6.1, la versiunile perfecționate ale familiei MCS-51 au fost introduse noi facilități "on chip". Una dintre cele mai importante este *zona de numărătoare programabile (PCA)*, care aduce un plus de flexibilitate în aplicațiile care necesită taskuri de timp variate, cum ar fi: comparare/captură evenimente, ieșiri de mare viteză, modularea în durată a impulsurilor (PWM), implementarea unor funcții de tip ceas de gardă (WDT) etc. Zona PCA se regăsește la versiunile 8xC51FA/FB/FC, iar seria 8xC51GB posedă chiar două astfel de zone. Evident, noile facilități presupun existența unor registre suplimentare în zona SFR. Pentru exemplificare se consideră familia 8xC51Fx, reprezentativă din punctul de vedere al noilor facilități introduse. Tabelul 6.17 prezintă harta zonei SFR, în care sunt evidențiate noile registre,

F8		CH	CCAP0H	CCAP1H	CCAP2H	CCAP3H	CCAP4H		FF
	B	00000000	xxxxxxx	xxxxxxx	xxxxxxx	xxxxxxx	xxxxxxx		
F0	00000000								
EB		CL	CCAP0L	CCAP1L	CCAP2L	CCAP3L	CCAP4L		F7
		00000000	xxxxxxx	xxxxxxx	xxxxxxx	xxxxxxx	xxxxxxx		
E0	ACC								E7
	00000000								
D8	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2	CCAPM3	CCAPM4		DF
	00x00000	00xx000	x0000000	x0000000	x0000000	x0000000	x0000000		
D0	PSW								
	00000000								
C8	T2CON	T2MOD	RCAP2L	RCAP2H	TL2	TH2			D7
	00000000	xxxxxx00	00000000	00000000	00000000	00000000			
C0									CF
B8	IP	SADEN							C7
	x0000000	00000000							
B0	P3								BF
	11111111								
A8	IE	SADDR						IPH	B7
	00000000	00000000						x0000000	
A0	P2								AF
	11111111								
98	SCON	SBUF							A7
	00000000	xxxxxxx							
90	P1								9F
	11111111								
88	TCON	TMOD	TL0	TL1	TH0	TH1			97
	00000000	00000000	00000000	00000000	00000000	00000000			
80	P0	SP	DPL	DPH					8F
	11111111	00000111	00000000	00000000					
							PCON		87
							00xx0000		

Tab.6.17. Harta zonei SFR și valorile după resetare la microcontrolerele 8xC51Fx

iar în tab.6.18 se prezintă semnificația acestora.

Mnemonica	Denumirea registrului special	Adresa SFR
IPH	Interrupt Priority High	B7h
SADEN	Serial Address Mask	B9h
SADDR	Serial Address	A9h
T2MOD	Timer/Counter 2 Mode Control	C9h
CCON	PCA Timer/Counter Control	D8h
CMOD	PCA Timer/Counter Mode	D9h
CCAPM0	PCA Timer/Counter Mode 0	DAh
CCAPM1	PCA Timer/Counter Mode 1	DBh
CCAPM2	PCA Timer/Counter Mode 2	DCh
CCAPM3	PCA Timer/Counter Mode 3	DDh
CCAPM4	PCA Timer/Counter Mode 4	DEh
CL	PCA Timer/Counter Low Byte	E9h
CH	PCA Timer/Counter High Byte	F9h
CCAP0L	PCA Compare/Capture Module 0 Low Byte	EAh
CCAP1L	PCA Compare/Capture Module 1 Low Byte	EBh
CCAP2L	PCA Compare/Capture Module 2 Low Byte	ECh
CCAP3L	PCA Compare/Capture Module 3 Low Byte	EDh
CCAP4L	PCA Compare/Capture Module 4 Low Byte	EEh
CCAP0H	PCA Compare/Capture Module 0 High Byte	FAh
CCAP1H	PCA Compare/Capture Module 1 High Byte	FBh
CCAP2H	PCA Compare/Capture Module 2 High Byte	FCh
CCAP3H	PCA Compare/Capture Module 3 High Byte	FDh
CCAP4H	PCA Compare/Capture Module 4 High Byte	FEh

Tab.6.18. Semnificația și adresele noilor registre de la familia 8xC51Fx

O altă facilitate s-a adăugat la Timerul 2, și anume: funcția de *numărător/temporizator reversibil (up/down timer/counter)*, regăsită la seriile 8xC51FA/FB/FC, 8xC52/54/58 și 8xC51GB. De asemenea, la toate aceste versiuni portului serial i-au fost adăugate noi capacități, precum: *detecția erorilor de încadrare (framing error detection)* și *recunoașterea automată a adresei (automatic address recognition)*, care înlesnesc detecția erorilor și comunicația multiprocesor.

Se observă că cele mai multe registre noi sunt pentru zona PCA. Întrucât componentele acestei zone pot genera întreruperi, a fost dezvoltat și sistemul de întreruperi, care la aceste microcontrolere posedă 7 surse cu prioritizare programabilă pe patru niveluri. Din acest motiv a fost necesar un registru suplimentar pentru priorități – IPH. Noile capacități ale portului serial au necesitat introducerea registrelor SADEN și SADDR, iar pentru Timerul 2 – T2MOD.

6.4.1. Zona de numărătoare programabile (PCA)

Zona de numărătoare programabile este formată dintr-un timer/counter de 16 biți și cinci module de comparare/captură de 16 biți. Schema bloc este prezentată în fig.6.38. Accesul la resursele zonei de numărătoare programabile se realizează prin Portul 1, conform cu tab.6.19.

Denumire pin	Funcția alternativă
P1.2	EC1 - Intrare externă pentru timerul/numărătorul PCA
P1.3	CEX0 - Intrare/ieșire pentru modulul 0 al PCA
P1.4	CEX1 - Intrare/ieșire pentru modulul 1 al PCA
P1.5	CEX2 - Intrare/ieșire pentru modulul 2 al PCA
P1.6	CEX3 - Intrare/ieșire pentru modulul 3 al PCA
P1.7	CEX4 - Intrare/ieșire pentru modulul 4 al PCA

Tab.6.19. Funcțiile alternative noi ale pinilor la MCS-51 cu PCA

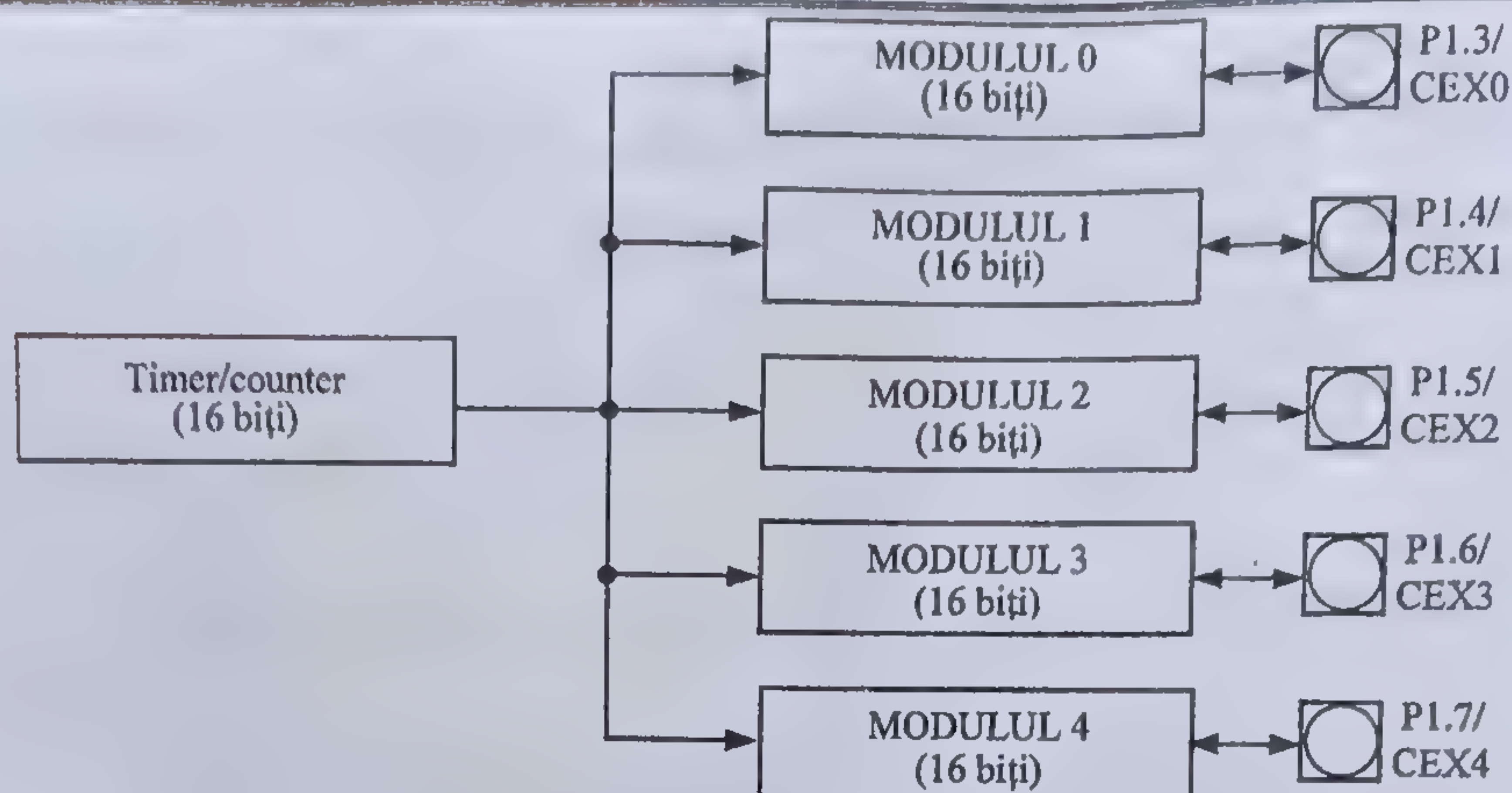


Fig.6.38. Structura zonei de număratoare programabile (PCA)

Timer/counter-ul este utilizat ca bază de timp comună pentru celelalte cinci module și este singurul timer al PCA. Este format din două numărătoare de 8 biți, CL și CH, cu adresele E9h și F9h, care pot fi atât înscrise cât și citite. În fig.6.39 este prezentată schema de funcționare a timer/counter-ului PCA.

Intrarea lui de tact poate fi selectată prin program să funcționeze în următoarele moduri:

a) Cu frecvența oscilatorului de tact divizată prin 12 (Modul 0). Registrul CL este incrementat în S5P2 al fiecărui ciclu mașină (v. §6.3.2). Dacă $f_{osc}=12\text{MHz}$, timerul se incrementează la fiecare $1\mu\text{s}$, iar dacă $f_{osc}=16\text{MHz}$ – la fiecare 750ns .

b) Cu frecvența oscilatorului de tact divizată prin 4 (Modul 1). Registrul CL este incrementat în S1P2, S3P2 și S5P2 a fiecărui ciclu mașină. Pentru $f_{osc}=12\text{MHz}$, timerul se incrementează la fiecare 330ns , iar dacă $f_{osc}=16\text{MHz}$ - la fiecare 250ns .

c) La depășirea de la Timerul 0 (Modul 2). Registrul CL se incrementează în S5P2 a ciclului mașină în care apare depășire la Timerul 0. În acest mod este posibilă programarea frecvenței de intrare a PCA.

d) Intrare externă prin linia ECI (Modul 3). Registrul CL este incrementat în prima dintre fazele S1P2, S3P2 și S5P2 ale fiecărui ciclu mașină de după detectarea unei tranziții $1 \rightarrow 0$ a

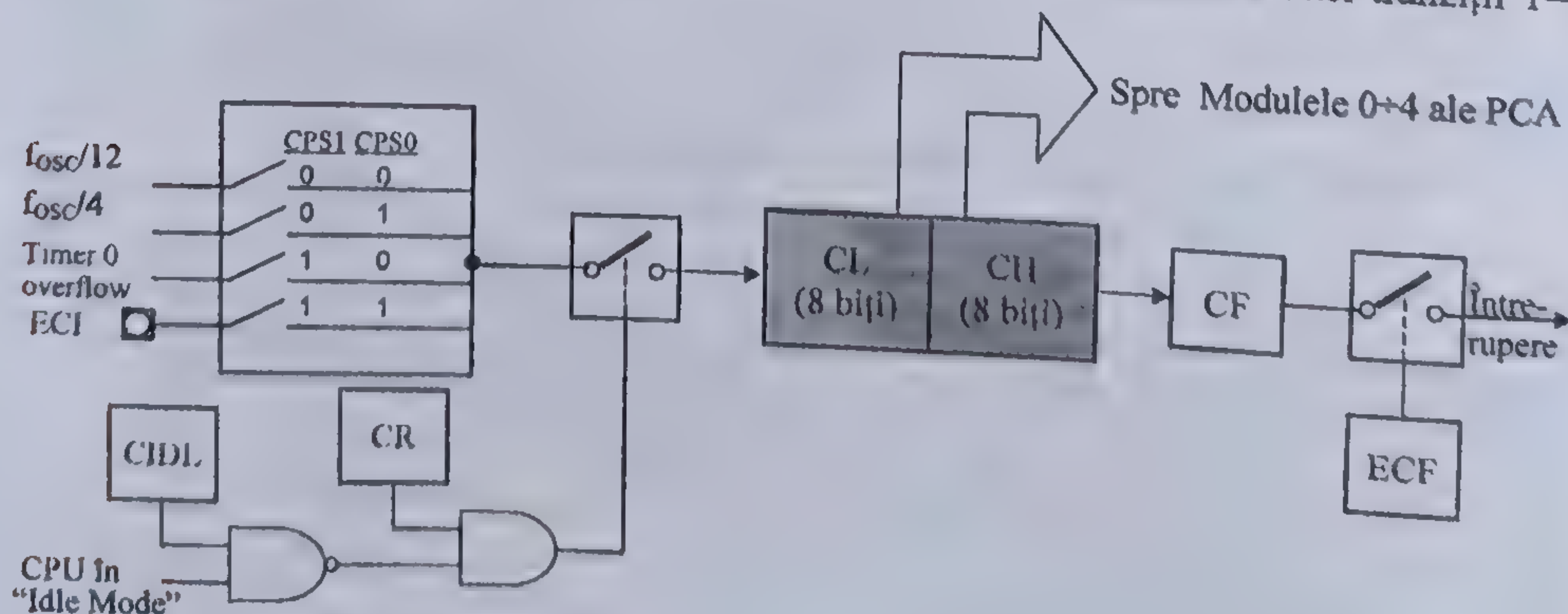


Fig.6.39. Schema de funcționare a Timer/Counter-ului PCA

semnalului pe intrarea ECI (P1.2). La $f_{osc}=12\text{MHz}$, incrementarea se face la $0,66\mu\text{s}$, iar la $f_{osc}=16\text{MHz}$ - la fiecare $0,5\mu\text{s}$. Registrul CH este incrementat la două perioade ale oscilatorului de la apariția depășirii în CL. Modurile de operare ale timer/counter-ului pot fi programate cu

(msb)					(lsb)			(C9h)
CIDL	WDTE	—	—	—	CPS1	CPS0	ECF	

Simbol	Funcția
CIDL	Controlul activității timer/counter-ului pe durata regimului inactiv (Counter IDLe control): CIDL=0 - timer/counter-ul continuă funcționarea pe durata “Idle Mode” CIDL=1 - timer/counter-ul suspendat pe durata “Idle Mode”.
WDTE	Activare WDT (WatchDog Timer Enable) la Modulul 4 al PCA.
—	Neutilizat, rezervat pentru dezvoltare *)
CPS1, 0	Biți de selecție intrare de numărare. <u>CPS1 CPS0. Selecție intrare PCA</u> <div> <div>00</div> <div>Tact intern, $f_{osc}/12$</div> </div> <div> <div>01</div> <div>Tact intern, $f_{osc}/4$</div> </div> <div> <div>10</div> <div>Depășire Timer 0</div> </div> <div> <div>11</div> <div>Tact extern, prin ECI (P1.2) -- intrarea maximă = $f_{osc}/8$</div> </div>
ECF	Validare întrerupere la depășire: la activarea bitului CF din CCON se va genera o întrerupere numai dacă ECF=1.

*) Obligatoriu se înscrie valoarea "0" logic. La citire se returnează orice valoare.

Fig.6.40. CMOD – Registrul modului de operare al timer/counter-ului PCA

biții CPS1 și CPS0 (Count Pulse Select) ai registrului CMOD, a cărei configurație este prezentată în fig.6.40. CMOD nu poate fi adresat la nivel de bit. Acest registru conține și bitul ECF (Enable Counter overFlow interrupt), care permite timer/counter-ului PCA să genereze întrerupere la depășire. Suplimentar, utilizatorul poate opta pentru deconectarea timer/counter-ului pe durata modului inactiv (Idle Mode), prin setarea bitului CIDL. Modulul 4 al PCA permite funcționarea ca ceas de gardă (WDT), activată prin bitul WDTE al registrului CMOD.

Din fig.6.39 se poate observa că la funcționarea timer/counter-ului PCA mai participă încă doi biți: CF și CR, care aparțin registrului de comandă al PCA, CCON (fig.6.41).

(msb)							(lsb)	
CF	CR	—	CCF4	CCF3	CCF2	CCF1	CCF0	(DSb)

Simbol	Funcția
CF	Fanion de întrerupere la depășire timer/counter. Este setat hardware la depășire, dar poate fi setat și software. CF semnalizează apariția unei întreruperi dacă bitul ECF din CMOD este setat. Poate fi resetat numai prin program.
CR	Bit de control a funcționării timer/counter-ului (Counter Run control bit). Se setează/resetează prin program pentru pornirea/oprirea timer/counter-ului.
—	Neutilizat, rezervat pentru dezvoltare *)
CCF4	Fanion întrerupere de la Modulul 4. Este setat HW; trebuie resetat SW.
CCF3	Fanion întrerupere de la Modulul 3. Este setat HW; trebuie resetat SW.
CCF2	Fanion întrerupere de la Modulul 2. Este setat HW; trebuie resetat SW.
CCF1	Fanion întrerupere de la Modulul 1. Este setat HW; trebuie resetat SW.
CCF0	Fanion întrerupere de la Modulul 0. Este setat HW; trebuie resetat SW.

*) Nu este permisă înscrierea cu "1" logic a acestui bit. La citire se returnează o valoare oarecare.

Fig.6.41. CCON - Registrul de controlul al timer/counter-ului PCA

Acest registru conține și fanioanele de depășire ale celor 5 module (CCF0÷CCF4). Toți indicatorii pot fi testați individual, întrucât registrul CCON este adresabil și la nivel de bit.

Modulele de comparare/captură pot fi programate fiecare într-unul din următoarele 6 moduri de funcționare: a) captură pe 16 biți, pe front crescător; b) captură pe 16 biți, pe front descrescător; c) captură pe 16 biți, pe ambele fronturi; d) timer software pe 16 biți; e) ieșire de mare viteză pe 16 biți; f) modulele a lățimii impulsurilor (PWM), pe 8 biți.



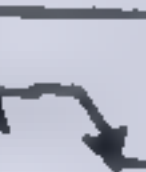
Când modulele de comparare/captură sunt programate într-unul din modurile: captură, timer software sau ieșire de mare viteză, acestea pot genera întreruperi, semnalizate prin biții CCF_n , ($n = 0 \div 4$) din registrul CCON (fig.6.41). Atât cele cinci module, cât și timerul PCA partajează într-o condiție "SAU logic" același vector de întreruperi: 0033h. Fiecare dintre module posedă câte un registru $CCAPM_n$, ($n=0 \div 4$) pentru selecția uneia dintre cele șase funcții posibile (fig.6.42). Combinațiile valide sunt prezentate în tab.6.20.

(msb)							(lsb)	
-	$ECOM_n$	$CAPP_n$	$CAPN_n$	MAT_n	TOG_n	PWM_n	$ECCF_n$	(DAh+DEh)

Simbol	Funcția
-	Neutilizat, rezervat pentru dezvoltare *)
$ECOM_n$	Activare comparator. $ECOM_n = 1$ activează funcția de comparare.
$CAPP_n$	$CAPP_n = 1$ validează captura pe front pozitiv.
$CAPN_n$	$CAPN_n = 1$ validează captura pe front negativ.
MAT_n	Potrivire (MATCH). $MAT_n = 1$ indică egalitatea conținutului timer/counter-ului cu cel al modulului n și determină setarea fanionului de întrerupere CCF_n din CCON.
TOG_n	Basculare (Toggle). $TOG_n = 1$ indică egalitatea conținutului timer/counter-ului cu cel al modulului n și determină bascularea ieșirilor CEX_n .
PWM_n	Setare mod de operare PWM la ieșirile CEX_n .
$ECCF_n$	Validare întrerupere CCF. Permite fanioanelor CCF_n din CCON să genereze întreruperi.

* Nu este permisă înscrierea cu "1" logic a acestui bit. La citire se returnează orice valoare.

Fig.6.42. $CCAPM_n$ – Registrul de mod al modulelor de comparare/captură din PCA

-	$ECOM_n$	$CAPP_n$	$CAPN_n$	MAT_n	TOG_n	PWM_n	$ECCF_n$	Funcția modulului
x	0	0	0	0	0	0	0	Nici o operație
x	x	1	0	0	0	0	x	Captură pe 16 biți, pe CEX_n 
x	x	0	1	0	0	0	x	Captură pe 16 biți, pe CEX_n 
x	x	1	1	0	0	0	x	Captură pe 16 biți, pe CEX_n 
x	1	0	0	1	1	0	x	Timer soft pe 16 biți
x	1	0	0	1	1	0	x	Ieșire de mare viteză pe 16 biți
x	1	0	0	0	0	1	0	PWM de 8 biți
x	1	0	0	1	x	0	x	WDT

Tab.6.20. Combinațiile valide pentru definirea funcțiilor Modulelor 0+4 ale PCA

Existența celor cinci module independente permite o creștere a numărului taskurilor ce pot fi tratate de un microcontroler, precum și o mai mare precizie decât în cazul utilizării timerelor. În plus, programul aferent și intervenția CPU sunt reduse la minimum. Ca atare, se poate spune că prezența PCA înlesnește și mai mult tratarea în timp real a evenimentelor comparativ cu versiunile standard 8051/8052.

6.4.2. Noile facilități ale Portului serial

Așa cum s-a menționat la începutul acestui capitol, noile facilități ale portului serial: *deteția erorilor de încadrare* (framing error detection) și *recunoașterea automată a adresei* (automatic address recognition), sporesc versatilitatea interfeței seriale privind detecția erorilor la recepție și comunicația multiprocesor.

6.4.2.1. Recunoașterea automată a adresei

În §6.3.4.2 s-a văzut că pentru aplicațiile care necesită o cooperare între microcontrolere, Portul serial poate funcționa în modul multiprocesor. Astfel, pentru implementarea unui protocol master-slave se folosește un al 9-lea bit de date, D8, care permite microcontrolerelor slave să facă distincție între un octet de adresă (D8=1) și unul de date (D8=0) transmise de microcontrolerul master. Inițial, sistemele slave setează SM2 în SCON, astfel încât să recepționeze numai octeți de tip adresă transmiși de master, după care sistemul slave adresat se reconfigurează (SM2=0) pentru a putea să recepționeze datele.

La seria 8051/8052/80C51, Portul serial al unui sistem slave generează întreruperi către UCP pentru toți octeții de tip adresă recepționați. Recunoașterea adresei se face prin program, în rutina de tratare a întreruperii de recepție serială, prin compararea adresei recepționate de la master cu adresa proprie. Astfel, toate procesoarele slave vor fi întrerupte de fiecare dată când sistemul master inițiază un dialog cu unul dintre sistemele slave. La noile familii, datorită facilității de *recunoaștere automată a adresei* (Automatic Address Recognition), numai slave-ul adresat își întrerupe funcționarea, compararea efectuându-se automat prin hardware, la nivelul blocului de recepție al Portului serial.

În acest mod, masterul poate stabili comunicația cu unul sau mai multe controlere slave, fără ca activitatea celorlalte procesoare slave să fie întreruptă, ceea ce elimină întreruperile inutile de recepție la nivelul sistemelor slave. Complexitatea rutinelor de întrerupere care implementează protocolul de acces la mediul de comunicație se reduce substanțial, mai ales atunci când la linia serială se conectează un număr mare de microcontrolere. În plus, facilitatea menționată poate fi utilizată în conjuncție cu modul "Idle" (v. §6.3.7.1), ceea ce permite o reducere considerabilă a puterii consumate de sistem.

Microcontrolerul master poate comunica selectiv cu un slave/grup de slave-uri prin folosirea unei *adrese specifice* (Given Address). Această adresă are o formă generalizată, prin introducerea unor biți indiferenți cu ajutorul unui octet-mască. De asemenea, master-ul poate adresa simultan toate procesoarele slave din sistem prin intermediul unei *adrese de difuzare* (Broadcast Address). Cele două noi tipuri de adrese se definesc, pentru fiecare microcontroler slave din sistem, prin intermediul noilor registre alocate portului serial în SFR: SADDR și SADEN (v. tab. 6.17 și 6.18). Registrul SADDR conține adresa individuală a unui slave, iar SADEN conține octetul-mască. Biții indiferenți ai adresei specifice se obțin în pozițiile biților "0" din SADEN.

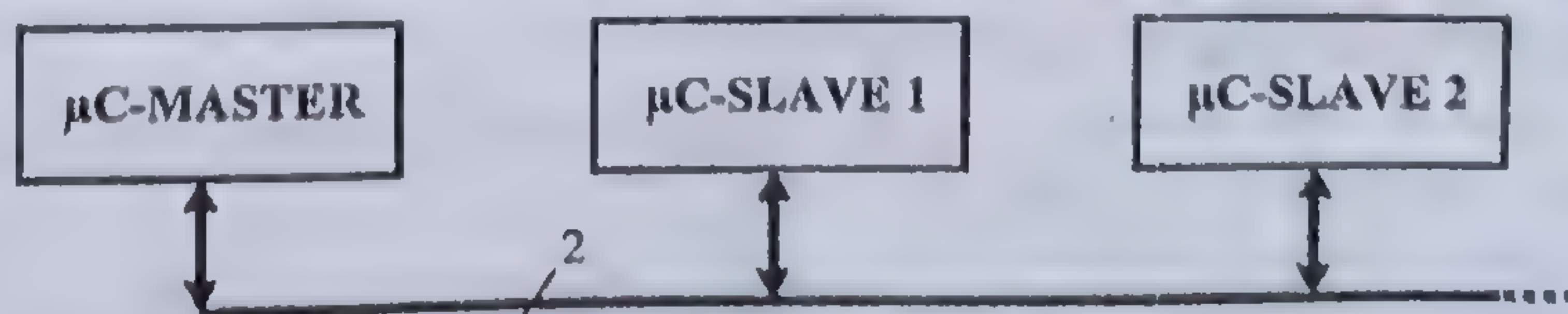


Fig.6.43. Comunicația serială multiprocesor cu protocol master-slave

Pentru exemplificare, se consideră un sistem cu trei microcontrolere: unul master și două slave (fig.6.43), interconectate prin intermediul porturilor seriale, similar structurilor

multiprocesor din seria 8051/8052/80C51 (v.fig.6.27). Pentru un număr redus de slave-uri este avantajos ca adresele individuale să fie stabilite prin semioctetul inferior din SADDR, semioctetul superior fiind "1111". Această alegere simplifică selectarea octetului mască ce trebuie depus în SADEN. Se consideră următoarele adrese individuale:

SLAVE 1: SADDR = 1111 0001; **SLAVE 2:** SADDR = 1111 0011

Octeții-mască se aleg astfel încât sistemele slave să poată fi adresate individual de către master, dar și simultan. Îndeplinirea condițiilor menționate se realizează dacă: bitul 0 (lsb) al *adresei specifice* pentru SLAVE 1 este logic indiferent (x), dar pentru SLAVE 2 este "1" logic. Similar, bitul 1 al *adresei specifice* este "0" logic pentru SLAVE 1 și "x" pentru SLAVE 2. Rezultă următoarele configurații pentru octeții-mască și pentru *adresele specifice* ale celor două slave-uri:

SLAVE 1	SLAVE 2
SADDR = 1111 0001	SADDR = 1111 0011
SADEN = 1111 1010	SADEN = 1111 1001
GIVEN = 1111 0x0x	GIVEN = 1111 0xx1

Astfel, masterul poate selecta numai SLAVE 1 printr-o adresă cu bitul 0 = 0 (spre ex. 1111 0000), iar pe SLAVE 2 printr-o adresă la care bitul 1 = 1 (de ex. 1111 0111). De asemenea, pentru a comunica cu ambele slave-uri simultan, adresa trebuie să aibă bitul 0 = 1 și bitul 1 = 0. De notat că, pe de altă parte, bitul 3 este indiferent pentru ambele slave-uri. Acest fapt permite utilizarea a două adrese pentru selecția ambelor controlere: 11110001 și 11110101.

În cazul în care este necesar să fie adăugat în sistem un al 3-lea slave, pentru acesta se impune ca bitul 2 = 0, ceea ce permite master-ului să poată comunica simultan cu primele două slave-uri folosind adresele de mai sus, fără a comunica și cu al treilea.

Dacă este necesar să se poată comunica simultan cu toate slave-urile se utilizează *adresa de difuzare* (Broadcast Address), care se formează simplu, printr-un SAU logic între SADDR și SADEN, în care zerourile pe același rang definesc "x" logic. De exemplu, adresa de difuzare rezultată pentru SLAVE 2 este următoarea:

$$\begin{array}{l} \text{SADDR} = 1111\ 0011 \\ \text{SADEN} = 1111\ 1001 \end{array} \quad \Rightarrow \quad \text{BROADCAST} = 1111\ 1x11$$

Se poate observa că această adresă este aceeași, indiferent pentru care slave se definește.

După resetarea microcontrolerelor, registrele SADDR și SADEN sunt inițializate cu 00h (v.tab.6.17), ceea ce conduce la definirea adreselor specifice și de difuzare sub forma xxxxxxxx (toți biții indiferenți). Acest fapt asigură familiei 8xC51Fx compatibilitate "în jos" cu celelalte componente ale familiei MCS-51 care nu posedă facilitatea de recunoaștere automată a adresei.

6.4.2.2. Detecția erorilor de încadrare

Această nouă facilitate a Portului serial permite controlerului receptor să verifice validitatea biților de stop (v.fig.6.24) în modurile 1, 2 și 3. Lipsa acestui bit din formatul cuvântului transmis conduce la erori de încadrare și poate fi cauzată fie de perturbații pe liniile seriale, fie de transmisia simultană a două porturi seriale.

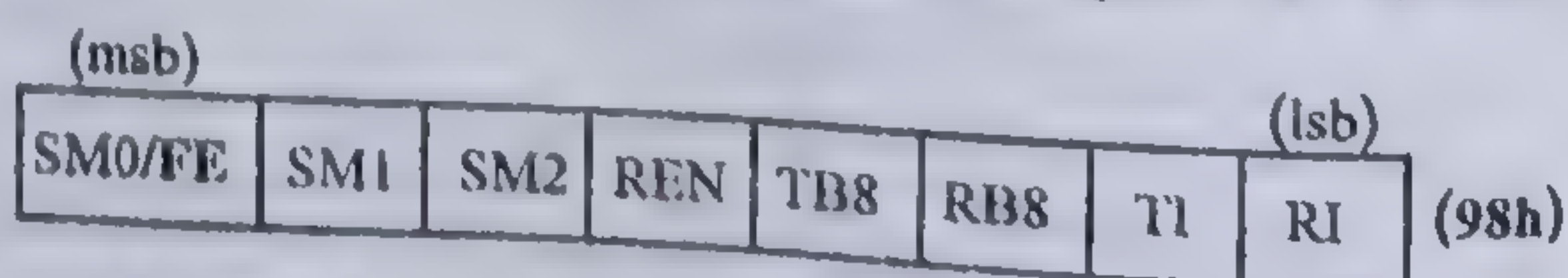


Fig.6.44. Registrul SCON la microcontrolerele cu posibilitatea de detectare a erorii de încadrare

Semnalizarea lipsei bitului de stop se face prin intermediul bitului SCON.7 din registrul SCON (v.fig.6.23), care în acest caz are funcția de *bit de eroare de încadrare* (FE-Framing Error bit), așa cum se arată în fig.6.44. Detectarea erorilor de încadrare se face prin testarea software a bitului FE după fiecare cuvânt recepționat. Odată setat, acest bit trebuie resetat prin program. Un bit de stop valid nu va reseta bitul FE. Deoarece FE partajează aceeași locație cu bitul SM0, pentru a determina care din acestea este accesat, în registrul PCON (v.fig.6.33) s-a introdus un nou bit - **SMOD0**, așa cum se vede în fig.6.45. Dacă SMOD0=0, atunci accesul la SCON.7 vizează SM0; dacă SMOD0=1, atunci accesul la SCON.7 este pentru FE. Bitul SMOD1 își păstrează aceeași funcționalitate definită în fig.6.33.

(msb)				(lsb)				
SMOD1	SMOD0	-	POF	GF1	GF0	PD	IDL	(87h)

Fig.6.45. Registrul PCON la microcontrolerele cu posibilitatea de detectare a erorii de încadrare

Bitul PCON.4 din PCON semnalizează o nouă facilitate în ceea ce privește controlul alimentării cu energie a microcontrolerelor din seriile 8xC51Fx, 8xC52/54/58 și 8xC51GB. Astfel, bitul **POF** (Power Off Flag) este setat prin hard atunci când V_{CC} este în jurul valorii nominale. Fanionul POF poate fi, de asemenea, setat sau resetat prin program, ceea ce permite utilizatorului să facă distincție între o resetare "la rece" (la conectarea tensiunii după o prealabilă deconectare) și una "la cald" (V_{CC} este aplicată dispozitivului, urmând ca acesta să revină din modul deconectat - v. §6.3.7.2).

6.5. Programarea familiei MCS-51

Toți membrii familiei MCS-51 execută același set de instrucțiuni, care este optimizat pentru aplicații de control. Această orientare este asigurată de o varietate de moduri rapide de adresare pentru accesarea memoriei RAM interne și care facilitează operații la nivel de octet asupra unor structuri de date de mici dimensiuni. De asemenea, setul de instrucțiuni conține un grup extins pentru operarea cu variabile de un bit, ca un tip distinct de date, ceea ce asigură manipularea directă a variabilelor booleene.

Formatul instrucțiunilor este variabil. Există 49 de instrucțiuni de 1 octet, 46 de 2 octeți și 16 de 3 octeți. Primul octet este întotdeauna codul operație (opcodul), iar următorii, dacă există, sunt operanzi sau adrese de operanzi de 1 sau 2 octeți. Faptul că 95 de instrucțiuni, din totalul de 111, sunt de 1 sau 2 octeți constituie un avantaj important, deoarece astfel codul executabil ocupă un spațiu relativ redus în memoria program. În tabelul 6.21 din §6.5.1 se va prezenta și formatul fiecărei instrucțiuni, cu numărul de octeți, semnificația lor și ordinea în care aceștia se amplasează în memorie.

Observație: Operanzii cu lungimea de 2 octeți sunt amplasați în memorie imediat după codul operației, mai întâi octetul mai semnificativ și apoi cel mai puțin semnificativ, adică tocmai invers decât la microprocesoarele Intel de uz general.

Modurile de adresare a operanzilor sunt variate și eficiente, în special pentru cei stocați în memoria internă de date, ceea ce constituie un alt atu al familiei MCS-51. Există 7 moduri de adresare de bază, definite pe scurt și exemplificate în continuare:

a) **adresarea directă**, în care operandul este specificat în instrucțiune printr-o adresă de 1 octet. Numai RAM-ul intern și zona SFR pot fi adresate direct.

MOV A, 30h ; în A se încarcă octetul din memoria internă de date, de la adresa 30h

b) *adresarea indirectă*, în care instrucțiunea precizează registrul care conține adresa operandului. Prin adresare indirectă pot fi specificați operanzi atât din RAM-ul intern, cât și din cel extern. Pentru adrese de RAM intern (de 8 biți) se pot utiliza registrele R0 sau R1 ale bancului de registre curent sau registrul indicator al vârfului stivei, SP. Pentru adrese de RAM extern (de 16 biți) poate fi utilizat numai registrul DPTR. În sintaxa instrucțiunii, numele acestor registre trebuie precedate de caracterul "@".

MOV @R0, A ; octetul din A este stocat în memoria internă de date, la adresa din R0

c) *adresarea cu registru (explicită)*, utilizează unul din registrele R0÷R7 ale bancului curent selectat în PSW. Specificarea registrului care conține operandul se realizează, ca și la microprocesoarele de uz general, printr-un câmp de 3 biți rezervat în opcodul instrucțiunii.

MOV A, R7 ; în A se încarcă octetul din registrul R7 (din bancul curent de registre)

d) *adresarea implicită (cu registru specific)*, în care nu se specifică explicit registrul care conține operandul. Ca și la microprocesoarele de uz general adresarea implicită folosește registrele speciale, în acest caz A, B, DPTR, sau fanionul de transport, C. Instrucțiunile care folosesc acest mod de adresare codifică registrul sau fanionul folosit implicit în opcodul instrucțiunii.

MUL AB ; înmulțește valoarea octetului din A cu cea a octetului din B; rezultatul în BA

e) *adresarea imediată*, în care operandul se află cuprins în instrucțiune, imediat după opcod. În instrucțiunile cu adresare imediată operandul trebuie precedat de caracterul "#".

MOV A, #30h ; în A se încarcă constanta 30h

f) *adresarea indexată*, în care drept registre index se utilizează DPTR sau PC. Acestea conțin adresa de bază, iar deplasamentul locației adresate se află implicit în acumulator, ceea ce permite codificarea instrucțiunilor cu adresare indexată pe un singur octet. Acest mod de adresare se folosește doar la adresarea operanzilor stocați în memoria program, facilitând procedura de consultare a tabelelor organizate în această memorie ("look-up tables"), precum și la o instrucțiune de salt indirect.

MOV A, @A+DPTR ; în A se încarcă octetul din memoria program de la adresa dată de A+DPTR

g) *adresarea relativă*, în care deplasamentul, de un octet cu semn (-128÷127), se adună la conținutul registrului PC pentru a calcula adresa la care va avea loc saltul. Acest mod de adresare se utilizează în instrucțiunile de ramificare, deplasamentul fiind indicat simbolic prin *rel*.

JNZ rel ; salt relativ la adresa instrucțiunii curente, dacă valoarea din A e diferită de 0

6.5.1. Setul de Instrucțiuni

În tab.6.21 este prezentat setul de instrucțiuni al familiei MCS-51, împărțit în 5 clase funcționale: instrucțiuni aritmetice, logice, de transfer, pentru manipularea variabilelor booleene și de ramificație. În afară de mnemonică și o scurtă descriere, tabelul conține și formatul instrucțiunilor. Toate codurile în hexa ale celor 111 instrucțiuni sunt prezentate în Anexa III. În ambele cazuri, notațiile utilizate au următoarele semnificații:

- Rn - unul din registrele R0÷R7 ale bancului de registre selectat
- direct - adresa de 8 biți a unei locații interne de date. Poate fi o locație din memoria RAM internă (0÷127) sau din zona SFR (128÷255).
- @Ri - locația de 8 biți a memoriei RAM interne (0÷255), adresată indirect prin registrele R0 sau R1 ale bancului de registre curent.

- #data** - constantă de 8 biți inclusă în instrucțiune
#data16 - constantă de 16 biți inclusă în instrucțiune
addr16 - adresă de 16 biți din spațiul de memorie program
addr11 - adresă de 11 biți dintr-o pagină de 2Ko de memorie program
rel - deplasament de 8 biți, cu semn (-128÷127)
bit - adresa unui bit dintr-o locație din zona RAM internă sau SFR, adresabilă pe bit

Tab.6.21

Nr. crt.	Mnemonica	Descrierea instrucțiunii	Formatul instrucțiunii	Nr. cicluri
Instrucțiuni aritmetice				
1	ADD A,Rn	Add register to accumulator $(A) \leftarrow (A) + (Rn),$ $n = 0 \div 7$	<div>0 0 1 0 1 r r r</div> <div>n</div>	1
2	ADD A,direct	Add direct byte to accumulator $(A) \leftarrow (A) + (direct)$	<div>0 0 1 0 0 1 0 1</div> <div>direct address</div>	1
3	ADD A,@Ri	Add indirect RAM to accumulator $(A) \leftarrow (A) + ((Ri)),$ $i = 0,1$	<div>0 0 1 0 0 1 1 i</div>	1
4	ADD A,#data	Add immediate data to accumulator $(A) \leftarrow (A) + data$	<div>0 0 1 0 0 1 0 0</div> <div>data</div>	1
5	ADDC A,Rn	Add register to A with carry flag $(A) \leftarrow (A) + (C) + (Rn),$ $n = 0 \div 7$	<div>0 0 1 1 1 r r r</div> <div>n</div>	1
6	ADDC A,direct	Add direct byte to A with carry flag $(A) \leftarrow (A) + (C) + (direct)$	<div>0 0 1 1 0 1 0 1</div> <div>direct address</div>	1
7	ADDC A,@Ri	Add indirect RAM to A with carry flag $(A) \leftarrow (A) + (C) + ((Ri)),$ $i = 0,1$	<div>0 0 1 1 0 1 1 i</div>	1
8	ADDC A,#data	Add immediate data to A with carry flag $(A) \leftarrow (A) + (C) + data$	<div>0 0 1 1 0 1 0 0</div> <div>data</div>	1
9	SUBB A,Rn	Subtract register from A with borrow $(A) \leftarrow (A) - (C) - (Rn),$ $n = 0 \div 7$	<div>1 0 0 1 1 r r r</div> <div>n</div>	1
10	SUBB A,direct	Subtract direct byte from A with borrow $(A) \leftarrow (A) - (C) - (direct)$	<div>1 0 0 1 0 1 0 1</div> <div>direct address</div>	1
11	SUBB A,@Ri	Subtract indirect RAM from A with borrow $(A) \leftarrow (A) - (C) - ((Ri)),$ $i = 0,1$	<div>1 0 0 1 0 1 1 i</div>	1
12	SUBB A,#data	Subtract immediate data from A with borrow $(A) \leftarrow (A) - (C) - data$	<div>1 0 0 1 0 1 0 0</div> <div>data</div>	1
13	INC A	Increment accumulator $(A) \leftarrow (A) + 1$	<div>0 0 0 0 0 1 0 0</div>	1

Tab.6.21 (continuare)

14	INC Rn	Increment register	$(Rn) \leftarrow (Rn) + 1,$ $n = 0+7$	<div>0 0 0 0 1 r r r</div> <div>n</div>	1
15	INC direct	Increment direct byte	$(direct) \leftarrow (direct) + 1$	<div>0 0 0 0 0 1 0 1</div> <div>direct address</div>	1
16	INC @Ri	Increment indirect RAM	$((Ri)) \leftarrow ((Ri)) + 1,$ $i = 0,1$	<div>0 0 0 0 0 1 1 i</div>	1
17	DEC A	Decrement accumulator	$(A) \leftarrow (A) - 1$	<div>0 0 0 1 0 1 0 0</div>	1
18	DEC Rn	Decrement register	$(Rn) \leftarrow (Rn) - 1,$ $n = 0+7$	<div>0 0 0 1 1 r r r</div> <div>n</div>	1
19	DEC direct	Decrement direct byte	$(direct) \leftarrow (direct) - 1$	<div>0 0 0 1 0 1 0 1</div> <div>direct address</div>	1
20	DEC @Ri	Decrement indirect RAM	$((Ri)) \leftarrow ((Ri)) - 1,$ $i = 0,1$	<div>0 0 0 1 0 1 1 i</div>	1
21	INC DPTR	Increment Data Pointer	$(DPTR) \leftarrow (DPTR) + 1$	<div>1 0 1 0 0 0 1 1</div>	2
22	MUL AB	Multiply A and B	$(B)(A) \leftarrow (A) \times (B)$	<div>1 0 1 0 0 1 0 0</div>	4
23	DIV AB	Divide A by B	$(A)(B) \leftarrow (A) / (B)$	<div>1 0 0 0 0 0 1 1</div>	4
24	DA A	Decimal adjust A	Dacă $\{ [(A_{3+0}) > 9] \vee [(AC)=1] \}$, atunci $(A_{3+0}) \leftarrow (A_{3+0}) + 6$. Dacă $\{ [(A_{7+4}) > 9] \vee [(C)=1] \}$, atunci $(A_{7+4}) \leftarrow (A_{7+4}) + 6$.	<div>1 1 0 1 0 1 0 0</div>	1
Instrucțiuni logice					
25	ANL A,Rn	AND register to A	$(A) \leftarrow (A) \wedge (Rn),$ $n = 0+7$	<div>0 1 0 1 1 r r r</div> <div>n</div>	1
26	ANL A,direct	AND direct byte to A	$(A) \leftarrow (A) \wedge (direct)$	<div>0 1 0 1 0 1 0 1</div> <div>direct address</div>	1
27	ANL A,@Ri	AND indirect RAM to A	$(A) \leftarrow (A) \wedge (Ri),$ $i = 0,1$	<div>0 1 0 1 0 1 1 i</div>	1
28	ANL A,#data	AND immediate data to A	$(A) \leftarrow (A) \wedge data$	<div>0 1 0 1 0 1 0 0</div> <div>data</div>	1
29	ANL direct,A	AND A to direct byte	$(direct) \leftarrow (direct) \wedge A$	<div>0 1 0 1 0 0 1 0</div> <div>direct address</div>	1

Tab.6.21 (continuare)

30	ANL <i>direct</i>,#<i>data</i>	AND immediate data to direct byte	$(direct) \leftarrow (direct) \wedge data$	<div>0 1 0 1 0 0 1 1</div> <div><i>direct</i> address</div> <div><i>data</i></div>	2
31	ORL A,<i>Rn</i>	OR register to A	$(A) \leftarrow (A) \vee (Rn),$ $n = 0 \div 7$	<div>0 1 0 0 1 <i>r r r</i></div> <div><i>n</i></div>	1
32	ORL A,<i>direct</i>	OR direct byte to A	$(A) \leftarrow (A) \vee (direct)$	<div>0 1 0 0 0 1 0 1</div> <div><i>direct</i> address</div>	1
33	ORL A,@<i>Ri</i>	OR indirect RAM to A	$(A) \leftarrow (A) \vee (Ri),$ $i = 0,1$	<div>0 1 0 0 0 1 1 <i>i</i></div>	1
34	ORL A,#<i>data</i>	OR immediate data to A	$(A) \leftarrow (A) \vee data$	<div>0 1 0 0 0 1 0 0</div> <div><i>data</i></div>	1
35	ORL <i>direct</i>,A	OR A to direct byte	$(direct) \leftarrow (direct) \vee A$	<div>0 1 0 0 0 0 1 0</div> <div><i>direct</i> address</div>	1
36	ORL <i>direct</i>,#<i>data</i>	OR immediate data to direct byte	$(direct) \leftarrow (direct) \vee data$	<div>0 1 0 0 0 0 1 1</div> <div><i>direct</i> address</div> <div><i>data</i></div>	2
37	XRL A,<i>Rn</i>	Exclusive OR register to A	$(A) \leftarrow (A) \oplus (Rn),$ $n = 0 \div 7$	<div>0 1 1 0 1 <i>r r r</i></div> <div><i>n</i></div>	1
38	XRL A,<i>direct</i>	Exclusive OR direct byte to A	$(A) \leftarrow (A) \oplus (direct)$	<div>0 1 1 0 0 1 0 1</div> <div><i>direct</i> address</div>	1
39	XRL A,@<i>Ri</i>	Exclusive OR indirect RAM to A	$(A) \leftarrow (A) \oplus (Ri),$ $i = 0,1$	<div>0 1 1 0 0 1 1 <i>i</i></div>	1
40	XRL A,#<i>data</i>	Exclusive OR immediate data to A	$(A) \leftarrow (A) \oplus data$	<div>0 1 1 0 0 1 0 0</div> <div><i>data</i></div>	1
41	XRL <i>direct</i>,A	Exclusive OR A to direct byte	$(direct) \leftarrow (direct) \oplus A$	<div>0 1 1 0 0 0 1 0</div> <div><i>direct</i> address</div>	1
42	XRL <i>direct</i>,#<i>data</i>	Exclusive OR immediate data to direct byte	$(direct) \leftarrow (direct) \oplus data$	<div>0 1 1 0 0 0 1 1</div> <div><i>direct</i> address</div> <div><i>data</i></div>	2
43	CLR A	Clear accumulator	$(A) \leftarrow 0$	<div>1 1 1 0 0 1 0 0</div>	1

Tab.6.21 (continuare)

44	CPL A	Complement accumulator	$(A) \leftarrow \neg(A)$	1 1 1 1 0 1 0 0	1
45	RL A	Rotate A left	$(A_{n+1}) \leftarrow (A_n), n = 0 \div 6,$ $(A_0) \leftarrow (A_7)$	0 0 1 0 0 0 1 1	1
46	RLC A	Rotate A left through carry flag	$(A_{n+1}) \leftarrow (A_n), n = 0 \div 6,$ $(A_0) \leftarrow (C), (C) \leftarrow (A_7)$	0 0 1 1 0 0 1 1	1
47	RR A	Rotate A right	$(A_n) \leftarrow (A_{n+1}), n = 0 \div 6,$ $(A_7) \leftarrow (A_0)$	0 0 0 0 0 0 1 1	1
48	RRC A	Rotate A right through carry flag	$(A_n) \leftarrow (A_{n+1}), n = 0 \div 6,$ $(A_7) \leftarrow (C), (C) \leftarrow (A_0)$	0 0 0 1 0 0 1 1	1
49	SWAP A	Swap nibble within A	$(A_{3-0}) \leftrightarrow (A_{7-4})$	1 1 0 0 0 1 0 0	1
Instrucțiuni de transfer					
50	MOV A,Rn	Move register to A	$(A) \leftarrow (Rn), n = 0 \div 7$	1 1 1 0 1 $\underbrace{r r r}_n$	1
51	MOV A,direct*	Move direct byte to A	$(A) \leftarrow (direct)$ * Instrucțiunea MOV A,ACC nu există	1 1 1 0 0 1 0 1 direct address	1
52	MOV A,@Ri	Move indirect RAM to A	$(A) \leftarrow ((Ri)), i = 0,1$	1 1 1 0 0 1 1 i	1
53	MOV A,#data	Move immediate data to A	$(A) \leftarrow data$	0 1 1 1 0 1 0 0 data	1
54	MOV Rn,A	Move A to register	$(Rn) \leftarrow (A), n = 0 \div 7$	1 1 1 1 1 $\underbrace{r r r}_n$	1
55	MOV Rn,direct	Move direct byte to register	$(Rn) \leftarrow (direct), n = 0 \div 7$	1 0 1 0 1 $\underbrace{r r r}_n$ direct address	2
56	MOV Rn,#data	Move immediate data to register	$(Rn) \leftarrow data, n = 0 \div 7$	0 1 1 1 1 $\underbrace{r r r}_n$ data	1
57	MOV direct,A	Move A to direct byte	$(direct) \leftarrow (A)$	1 1 1 1 0 1 0 1 direct address	1
58	MOV direct,Rn	Move register to direct byte	$(direct) \leftarrow (Rn), n = 0 \div 7$	1 0 0 0 1 $\underbrace{r r r}_n$ direct address	2
59	MOV direct,direct	Move direct byte to direct byte	$(direct) \leftarrow (direct)$	1 0 0 0 0 1 0 1 direct addr. (src.) direct addr. (dest.)	2

Tab.6.21 (continuare)

60	MOV <i>direct</i>,@R<i>i</i>	Move indirect RAM to direct byte	$(direct) \leftarrow ((Ri)), i = 0,1$	<div>1 0 0 0 0 1 1 <i>i</i></div> <div><i>direct address</i></div>	2
61	MOV <i>direct</i>,#<i>data</i>	Move immediate data to direct byte	$(direct) \leftarrow data$	<div>0 1 1 1 0 1 0 1</div> <div><i>direct address</i></div> <div><i>data</i></div>	2
62	MOV @R<i>i</i>,A	Move A to indirect RAM	$((Ri)) \leftarrow (A), i = 0,1$	<div>1 1 1 1 0 1 1 <i>i</i></div>	1
63	MOV @R<i>i</i>,<i>direct</i>	Move direct byte to indirect RAM	$((Ri)) \leftarrow (direct), i = 0,1$	<div>1 0 1 0 0 1 1 <i>i</i></div> <div><i>direct address</i></div>	2
64	MOV @R<i>i</i>,#<i>data</i>	Move immediate data to indirect RAM	$((Ri)) \leftarrow data, i = 0,1$	<div>0 1 1 1 0 1 1 <i>i</i></div> <div><i>data</i></div>	1
65	MOV DPTR,#<i>data16</i>	Load Data Pointer with a 16-bit constant	$(DPTR) \leftarrow data16$	<div>1 0 0 1 0 0 0 0</div> <div>HIGH (<i>addr16</i>)</div> <div>LOW (<i>addr16</i>)</div>	2
66	MOVC A,@A+DPTR	Move code byte relative to DPTR to A	$(A) \leftarrow ((A)+(DPTR))$	<div>1 0 0 1 0 0 1 1</div>	2
67	MOVC A,@A+PC	Move code byte relative to PC to A	$(PC) \leftarrow (PC)+1$ $(A) \leftarrow ((A) + (PC))$	<div>1 0 0 0 0 0 1 1</div>	2
68	MOVX A,@R<i>i</i>	Move external RAM (8-bit address) to A	$(A) \leftarrow ((Ri)), i = 0,1$	<div>1 1 1 0 0 0 1 <i>i</i></div>	2
69	MOVX A,@DPTR	Move external RAM (16-bit address) to A	$(A) \leftarrow ((DPTR))$	<div>1 1 1 0 0 0 0 0</div>	2
70	MOVX @R<i>i</i>,A	Move A to external RAM (8-bit address)	$((Ri)) \leftarrow (A), i = 0,1$	<div>1 1 1 1 0 0 1 <i>i</i></div>	2
71	MOVX @DPTR,A	Move A to external RAM (16-bit addr.)	$((DPTR) \leftarrow (A))$	<div>1 1 1 1 0 0 0 0</div>	2
72	PUSH <i>direct</i>	Push direct byte onto stack	$(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (direct)$	<div>1 1 0 0 0 0 0 0</div> <div><i>direct address</i></div>	2
73	POP <i>direct</i>	Pop direct byte from stack	$(direct) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$	<div>1 1 0 1 0 0 0 0</div> <div><i>direct address</i></div>	2
74	XCH A,R<i>n</i>	Exchange register with A	$(A) \leftrightarrow (Rn), n = 0-7$	<div>1 1 0 0 1 <i>r r r</i></div> <div><i>n</i></div>	1

Tab.6.21 (continuare)

75	XCH A,direct	Exchange direct byte with A	$(A) \leftrightarrow (direct)$	<div>1 1 0 0 0 1 0 1</div> <div>direct address</div>	1
76	XCH A,@Ri	Exchange indirect RAM with A	$(A) \leftrightarrow ((Ri)), i = 0,1$	<div>1 1 0 0 0 1 1 i</div>	1
77	XCHD A,@Ri	Exchange low-order Digit indirect RAM with A	$(A_{3,0}) \leftrightarrow ((Ri)_{3,0}), i = 0,1$	<div>1 1 0 1 0 1 1 i</div>	1
Instrucțiuni pentru manipularea variabilelor booleene					
78	CLR C	Clear carry flag	$(C) \leftarrow 0$	<div>1 1 0 0 0 0 1 1</div>	1
79	CLR bit	Clear direct bit	$(bit) \leftarrow 0$	<div>1 1 0 0 0 0 1 0</div> <div>bit address</div>	1
80	CPL C	Complement carry	$(C) \leftarrow \neg(C)$	<div>1 0 1 1 0 0 1 1</div>	1
81	CPL bit	Complement direct bit	$(bit) \leftarrow \neg(bit)$	<div>1 0 1 1 0 0 1 0</div> <div>bit address</div>	1
82	SETB C	Set carry flag	$(C) \leftarrow 1$	<div>1 1 0 1 0 0 1 1</div>	1
83	SETB bit	Set direct bit	$(bit) \leftarrow 1$	<div>1 1 0 1 0 0 1 0</div> <div>bit address</div>	1
84	ANL C,bit	AND direct bit to carry flag	$(C) \leftarrow (C) \wedge (bit)$	<div>1 0 0 0 0 0 1 0</div> <div>bit address</div>	2
85	ANL C,/bit	AND complement of direct bit to carry	$(C) \leftarrow (C) \wedge \neg(bit)$	<div>1 0 1 1 0 0 0 0</div> <div>bit address</div>	2
86	ORL C,bit	OR direct bit to carry flag	$(C) \leftarrow (C) \vee (bit)$	<div>0 1 1 1 0 0 1 0</div> <div>bit address</div>	2
87	ORL C,/bit	OR complement of direct bit to carry	$(C) \leftarrow (C) \vee \neg(bit)$	<div>1 0 1 0 0 0 0 0</div> <div>bit address</div>	2
88	MOV C,bit	Move direct bit to carry flag	$(C) \leftarrow (bit)$	<div>1 0 1 0 0 0 1 0</div> <div>bit address</div>	1
89	MOV bit,C	Move carry flag to direct bit	$(bit) \leftarrow (C)$	<div>1 0 0 1 0 0 1 0</div> <div>bit address</div>	2

Tab.6.21 (continuare)

Instrucțiuni de ramificare					
90	ACALL <i>addr11</i>	Absolute subroutine call	$(PC) \leftarrow (PC) + 2$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{7+0})$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{15+8})$ $(PC_{10+0}) \leftarrow addr11$	$a_{10} a_9 a_8 1 0 0 0 1$ $a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0$	2
91	LCALL <i>addr16</i>	Long subroutine call	$(PC) \leftarrow (PC) + 3$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{7+0})$ $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (PC_{15+8})$ $(PC) \leftarrow addr16$	$0 0 0 1 0 0 1 0$ HIGH (<i>addr16</i>) LOW (<i>addr16</i>)	2
92	RET	Return from subroutine	$(PC_{15+8}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$ $(PC_{7+0}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$	$0 0 1 0 0 0 1 0$	2
93	RETI	Return from interrupt	$(PC_{15+8}) \leftarrow (SP)$ $(SP) \leftarrow (SP) - 1$ $(PC_{7+0}) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$	$0 0 1 1 0 0 1 0$	2
94	AJMP <i>addr11</i>	Absolute jump address	$(PC) \leftarrow (PC) + 2$ $(PC_{10+0}) \leftarrow addr11$	$a_{10} a_9 a_8 0 0 0 1$ $a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0$	2
95	LJMP <i>addr16</i>	Long jump	$(PC) \leftarrow addr16$	$0 0 0 0 0 0 1 0$ HIGH (<i>addr16</i>) LOW (<i>addr16</i>)	2
96	SJMP <i>rel</i>	Short jump (relative address)	$(PC) \leftarrow (PC) + 2$ $(PC) \leftarrow (PC) + rel$	$1 0 0 0 0 0 0 0$ <i>rel. address</i>	2
97	JMP @A+DPTR	Jump indirect relative to the DPTR	$(PC) \leftarrow (A) + (DPTR)$	$0 1 1 1 0 0 1 1$	2
98	JZ <i>rel</i>	Jump if A is zero	$(PC) \leftarrow (PC) + 2$ Dacă $(A) = 0$, atunci $(PC) \leftarrow (PC) + rel$	$0 1 1 0 0 0 0 0$ <i>rel. address</i>	2
99	JNZ <i>rel</i>	Jump if A is not zero	$(PC) \leftarrow (PC) + 2$ Dacă $(A) \neq 0$, atunci $(PC) \leftarrow (PC) + rel$	$0 1 1 1 0 0 0 0$ <i>rel. address</i>	2
100	JC <i>rel</i>	Jump if carry flag is set	$(PC) \leftarrow (PC) + 2$ Dacă $(C) = 1$, atunci $(PC) \leftarrow (PC) + rel$	$0 1 0 0 0 0 0 0$ <i>rel. address</i>	2
101	JNC <i>rel</i>	Jump if carry flag is not set	$(PC) \leftarrow (PC) + 2$ Dacă $(C) = 1$, atunci $(PC) \leftarrow (PC) + rel$	$0 1 0 1 0 0 0 0$ <i>rel. address</i>	2

Tab.6.21 (continuare)

102	JB <i>bit,rel</i>	Jump if direct bit is set	$(PC) \leftarrow (PC) + 3$ Dacă (<i>bit</i>) = 1, atunci $(PC) \leftarrow (PC) + rel$	<div>0 0 1 0 0 0 0 0</div> <div><i>bit</i> address</div> <div><i>rel.</i> address</div>	2
103	JNB <i>bit,rel</i>	Jump if direct bit is not set	$(PC) \leftarrow (PC) + 3$ Dacă (<i>bit</i>) = 0, atunci $(PC) \leftarrow (PC) + rel$	<div>0 0 1 1 0 0 0 0</div> <div><i>bit</i> address</div> <div><i>rel.</i> address</div>	2
104	JBC <i>bit,rel</i>	Jump if direct bit is set and clear bit	$(PC) \leftarrow (PC) + 3$ Dacă (<i>bit</i>) = 1, atunci (<i>bit</i>) \leftarrow 0 și $(PC) \leftarrow (PC) + rel$	<div>0 0 0 1 0 0 0 0</div> <div><i>bit</i> address</div> <div><i>rel.</i> address</div>	2
105	CJNE <i>A,direct,rel</i>	Compare direct byte to A and jump if not equal	$(PC) \leftarrow (PC) + 3$ Dacă (<i>A</i>) \neq (<i>direct</i>), atunci $(PC) \leftarrow (PC) + rel$ Dacă (<i>A</i>) < (<i>direct</i>), atunci (<i>C</i>) \leftarrow 1, altfel (<i>C</i>) \leftarrow 0.	<div>1 0 1 1 0 1 0 1</div> <div><i>direct</i> address</div> <div><i>rel.</i> address</div>	2
106	CJNE <i>A,#data,rel</i>	Compare immediate byte to A and jump if not equal	$(PC) \leftarrow (PC) + 3$ Dacă (<i>A</i>) \neq <i>data</i> , atunci $(PC) \leftarrow (PC) + rel$. Dacă (<i>A</i>) < <i>data</i> , atunci (<i>C</i>) \leftarrow 1, altfel (<i>C</i>) \leftarrow 0.	<div>1 0 1 1 0 1 0 0</div> <div><i>data</i></div> <div><i>rel.</i> address</div>	2
107	CJNE <i>Rn,#data,rel</i>	Compare immediate byte to register and jump if not equal	$(PC) \leftarrow (PC) + 3$ Dacă (<i>Rn</i>) \neq <i>data</i> , atunci $(PC) \leftarrow (PC) + rel$. Dacă (<i>Rn</i>) < <i>data</i> , atunci (<i>C</i>) \leftarrow 1, altfel (<i>C</i>) \leftarrow 0.	<div>1 0 1 1 1 <i>r r r</i></div> <div><i>data</i></div> <div><i>rel.</i> address</div>	2
108	CJNE <i>@Ri,#data,rel</i>	Compare immediate byte to indirect and jump if not equal	$(PC) \leftarrow (PC) + 3$ Dacă (<i>Ri</i>) \neq <i>data</i> , atunci $(PC) \leftarrow (PC) + rel$ Dacă (<i>Ri</i>) < <i>data</i> , atunci (<i>C</i>) \leftarrow 1, altfel (<i>C</i>) \leftarrow 0.	<div>1 0 1 1 0 1 1 <i>i</i></div> <div><i>data</i></div> <div><i>rel.</i> address</div>	2
109	DJNZ <i>Rn,rel</i>	Decrement register and jump if not zero	$(PC) \leftarrow (PC) + 2$ (<i>Rn</i>) \leftarrow (<i>Rn</i>) - 1, <i>n</i> = 0+7. Dacă (<i>Rn</i>) \neq 0, atunci $(PC) \leftarrow (PC) + rel$	<div>1 1 0 1 1 <i>r r r</i></div> <div><i>rel.</i> address</div>	2
110	DJNZ <i>direct,rel</i>	Decrement direct byte and jump if not zero	$(PC) \leftarrow (PC) + 2$ (<i>direct</i>) \leftarrow (<i>direct</i>) - 1 Dacă (<i>direct</i>) \neq 0, atunci $(PC) \leftarrow (PC) + rel$	<div>1 1 0 1 0 1 0 1</div> <div><i>direct</i> address</div> <div><i>rel.</i> address</div>	2
111	NOP	No operation	$(PC) \leftarrow (PC) + 1$	<div>0 0 0 0 0 0 0 0</div>	1

Tabelul 6.21 nu conține nici o referire la indicatorii de condiție afectați. Un număr relativ mic de tipuri de instrucțiuni afectează doar o parte dintre acești indicatori, așa cum reiese și din tab.6.22. De asemenea, operațiile directe asupra octetului cu adresa D0h din zona SFR (adică registrul PSW) sau asupra biților din zona D0h+D7h afectează fanioanele de condiții corespunzătoare, dacă se ține seama că fanioanele de condiții pot fi adresate direct pe bit, având următoarele adrese: OV=D2h, AC=D6h, C=D7h.

Tab.6.22

Instrucțiune	C	OV	AC
Instrucțiuni care afectează indirect indicatorii de condiție			
ADD	!	!	!
ADC	!	!	!
SUBB	!	!	!
MUL	0	!	-
DIV	0	!	-
RRC	!	-	-
RLC	!	-	-
Instrucțiuni care afectează direct indicatorii de condiție			
SETB C	1	-	-
CLR C	0	-	-
ANL C, bit	!	-	-
ANL C, /bit	!	-	-
ORL C, bit	!	-	-
ORL C, /bit	!	-	-
MOV C, bit	!	-	-
CJNE	!	-	-
0 : resetat; 1 : setat; × : afectat în funcție de rezultat; - : neafectat			

6.5.2. Programarea în limbaj de asamblare

Ca și la sistemele cu microprocesoare de uz general, scrierea programelor sursă în limbaj de asamblare necesită atât cunoașterea setului de instrucțiuni, cât și a regulilor sintactice și a directiveiilor specifice programelor de asamblare automată (macro-asamblare).

Pentru microcontrolerele din familia MCS-51, firma Intel a produs macro-asamblorul ASM51, care stă la baza majorității programelor de acest tip existente în prezent. Dintre acestea se pot menționa produsele firmelor Keil/Franklin Software, IAR Systems, Archimedes Software, Avocet Systems, BSO/Tasking ș.a. [52]. Acestea sunt de obicei incluse în medii software integrate de dezvoltare a aplicațiilor, împreună cu compilatoare C, editoare de legături, simulatoare, depanatoare și alte programe utilitare. Versiuni demonstrative și de evaluare sunt puse la dispoziție și pot fi încărcate, instalate și testate gratuit de pe site-urile Internet ale unor firme cum sunt Keil Software Inc. (<http://www.keil.com>) sau Franklin Software Inc. (<http://www.fsinc.com>).

În cele ce urmează se prezintă principalele directive de asamblare și reguli sintactice de scriere a fișierelor sursă, așa cum sunt ele delinite de macroasamblorul ASM51.

Din punct de vedere logic, un program poate avea acces la mai multe tipuri de segmente de memorie: de cod (CODE), de date externe (DATA), de cod și date externe sau comune (XDATA), de date interne (IDATA), de biți în memoria RAM internă (BIT). Un astfel de segment poate fi absolut (CSEG, DSEG, XSEG, ISEG, BSEG), având adresa de încărcare

stabilită de programator la scrierea programului sursă, sau poate fi relocabil (RSEG), caz în care adresa unde se va găsi segmentul la momentul execuției va fi fixată ulterior, în mod automat, de către editorul de legături.

Exemplu:

?RELCOD	SEGMENT	CODE ; ?RELCOD este un segment de cod relocabil
RSEG ?RELCOD		; secvența de instrucțiuni care urmează face parte din segmentul de ; cod relocabil ?RELCOD
.....		
CSEG AT 001Bh		; forțează amplasarea secvenței de cod care urmează la adresa 001Bh ; (segment de cod absolut)
.....		
?RELDATA	SEGMENT	DATA
RSEG ?RELDATA		; urmează secvența de pseudoinstrucțiuni de amplasare statică a datelor ; în memoria externă
.....		

ORG exp - această pseudoinstrucțiune se folosește pentru a forța amplasarea unor secvențe de instrucțiuni la anumite deplasamente fixe față de adresa de început a modului de program curent. Expresia **exp** precizează deplasamentul (offset-ul) adresei de memorie începând cu care vor fi amplasate instrucțiunile care urmează după **ORG** (ORiGine), față de începutul modului relocabil curent (0000h). În cadrul operației de asamblare, contorul de program gestionat de asamblor pentru segmentul curent va fi încărcat cu valoarea expresiei din câmpul operand, **exp**.

const EQU exp (EQUate) - atribuie în mod *permanent* constantei, cu numele simbolic **const**, valoarea dată de expresia **exp** (o expresie numerică, logică sau un alt nume simbolic).

cstemp SET exp - atribuie în mod *temporar* constantei, cu numele simbolic **cstemp**, valoarea dată de expresia **exp**. Singura deosebire față de **EQU** este aceea că simbolul definit cu directiva **SET** se poate redefini ulterior, în cadrul aceluiași fișier sursă, ori de câte ori este nevoie. Asamblorul substituie numele simbolice **const** sau **cstemp** cu valorile atribuite cu ajutorul directivelor **EQU** și **SET**; de aceea, ele trebuie să preceadă orice utilizare a numelor simbolice respective.

[etich:] **DB listă** (Define Byte) - definește valoarea numelui simbolic **etich** [opțional] ca fiind adresa de început a unei zone de date alocată static în memoria externă, organizată pe octet și inițializată cu valorile elementelor din listă. Acestea pot fi valori numerice (0÷255), constante simbolice (definite anterior cu **EQU** sau **SET**), șiruri de caractere ASCII între ghilimele simple, expresii aritmetice și logice, toate separate prin virgulă. Directiva **DB** se utilizează de regulă pentru a defini date de 1 octet, constante (în memoria de cod) sau variabile inițializate static (numai în memoria comună).

[etich:] **DW lista** (Define Word) - la fel ca **DB**, dar zona de memorie alocată este organizată pe cuvinte de 2 octeți. În listă se află de obicei valori numerice (0÷65535) sau adrese simbolice (etichete) de instrucțiuni ori de date definite în fișierul sursă; octetul mai semnificativ se memorează la adresa mai mică (invers decât la microprocesoarele de uz general).

[etich:] **DBIT lista** (Define Bit) - la fel ca **DB**, dar "zona" de memorie alocată este de 1 bit (v.fig.6.25a). În listă se află de obicei valori binare, 0 sau 1.

[etich:] **DS exp** (Define Storage) - definește valoarea numelui simbolic **etich** [opțional] ca fiind adresa de început a unei zone de memorie RAM (de date sau comună), alocată static, de dimensiune egală cu valoarea numerică a expresiei **exp**, dar neinițializată. În câmpul operand

exp se poate afla un număr, o constantă simbolică definită anterior (cu EQU sau SET) sau o expresie aritmetică.

PUBLIC lista, **EXTRN** lista - aceste directive se utilizează atunci când două sau mai multe fișiere sursă se referă la o aceeași adresă simbolică (etichetă) de cod sau de date. Unicitatea presupune că aceasta este definită o singură dată (prin etich:), numai într-unul din fișierele sursă; în acel fișier se utilizează directiva **PUBLIC**, pentru a o face astfel cunoscută (publică) în exterior, iar **EXTRN** trebuie utilizată în restul fișierelor, care conțin referiri la acea adresă simbolică. Lista din câmpul de operand poate conține mai multe astfel de simboluri, separate prin virgulă.

O categorie distinctă o constituie directivele de *asamblare condiționată*. O secvență de asamblare condiționată are următoarea formă:

```
IF arg
    secvența 1
[ELSE|ELSEIF
    secvența 2]
ENDIF
```

În câmpul arg se poate afla o expresie logică sau o expresie aritmetică. În cazul în care expresia logică este adevărată, respectiv expresia aritmetică este evaluată la o valoare diferită de 0 - se assemblează secvența 1, altfel - se assemblează secvența 2. Directiva **ELSE|ELSEIF** este opțională, dar **ENDIF** încheie întotdeauna secvența de asamblare condiționată. Operatorii logici care pot fi utilizați sunt: EQ (Equal), NE (Not Equal), LT (Less Than), LE (Less than or Equal), GT (Greater Than), GE

(Greater than or Equal).

END - indică sfârșitul fișierului sursă, fiind delimitatorul său final. Detectarea sa de către asamblor va determina sfârșitul unei faze (tregeri) a asamblării.

La scrierea programelor în limbaj de asamblare trebuie respectate anumite reguli sintactice, specifice limbajelor simbolice, definite de programul asamblor. Ca și majoritatea asambloarelor, ASM51 acceptă fișiere sursă formate din linii de text ASCII având cele patru câmpuri specifice, și anume: *eticheta*, *cod*, *operand* și *comentariu*. Câmpurile sunt despărțite prin delimitatori de câmp, care pot fi blankuri sau caractere speciale. Folosind notația BNF (v. §1.1.1.4), pentru o linie sursă în limbaj de asamblare se poate scrie:

<INSTR> ::= <ETICHETA> DC<COD>DC<OPERAND>DC<COMENTARIU>.

unde prin DC s-au specificat delimitatorii de câmp. Asamblorul trebuie să recunoască fiecare câmp și să-l interpreteze în mod corespunzător. Câmpurile COD și OPERAND corespund unei instrucțiuni din setul cu care este dotată familia MCS-51 sau unei pseudoinstrucțiuni specifice asamblorului.

Eticheta marchează numele simbolic al unei constante, al unei adrese sau al unui segment relocabil. Într-o instrucțiune nu este obligatorie prezența etichetei, aceasta fiind folosită în funcție de necesitățile programatorului. Obişnuit, o etichetă începe cu o literă sau cu unul din caracterele speciale '?' sau '_'. După ultimul caracter al etichetei trebuie să urmeze caracterul ":", dacă ea marchează o adresă de memorie (*delimitatorul de câmp*). Eticheta poate fi definită astfel:

<ETICHETA> ::= <SIMBOL>

<SIMBOL> ::= <LITERA>|<LITERA><CARACTER>|<LITERA><SIMBOL>

Nu pot fi folosite ca etichete mnemonicele operațiilor sau pseudoinstrucțiunile și nici numele registrelor interne ale procesorului.

Codul este format numai din numele instrucțiunilor sau al pseudoinstrucțiunilor acceptate de asamblor. Asamblorul nu distinge între litere mici și litere mari.

<COD> := MOVX|ADDC|PUSH|...|ORG|...|DBIT

Operandul indică adrese sau date necesare câmpului COD. În funcție de instrucțiune, operandul poate lipsi sau poate conține un număr finit de argumente, separate prin virgulă.

$\langle \text{OPERAND} \rangle ::= | \langle \text{ARGUMENT} \rangle | \langle \text{ARGUMENT} \rangle, \dots, \langle \text{ARGUMENT} \rangle$

În câmpul de argument se pot afla nume de registre, constante numerice sau simbolice, etichete, expresii aritmetice sau logice):

$\langle \text{ARGUMENT} \rangle ::= \langle \text{CONSTANTA} \rangle | \langle \text{SIMBOL} \rangle |$
 $\langle \text{SIMBOL} \rangle \langle \text{OP} \rangle \langle \text{ARGUMENT} \rangle |$
 $\langle \text{CONSTANTA} \rangle \langle \text{OP} \rangle \langle \text{ARGUMENT} \rangle,$

unde OP reprezintă un operator : $\langle \text{OP} \rangle ::= + | - | * | /$

Câmpul operand poate specifica: un registru, o adresă de SFR, o constantă numerică sau simbolică de 1 sau 2 octeți, o expresie aritmetică sau o etichetă dintr-un segment de cod sau de date. Când sunt doi operanzi, primul operand specifică destinația operației, iar al doilea specifică sursa; cei doi operanzi se separă prin virgulă. Operanzii de tip constantă pot fi exprimați în următoarele moduri:

1. O constantă zecimală precedată de caracterul '#'.

Etichetă Cod Operand Comentariu

et1: MOV A,#58 ; Inițializează acumulatorul cu constanta $58_{10} = 3 \times 16 + 10 = 3A_{16}$.

2. O constantă hexazecimală, care de asemenea trebuie să înceapă cu '#' și cu o cifră (0÷9) și să se termine cu litera 'h' (sau 'H').

et2: MOV DPTR,#0FF00h ; Poziționează pointerul de date externe la adresa $FF00_{16}$.

3. O constantă octală care trebuie să înceapă cu '#' și să se termine cu litera 'q' ('Q').

MOV A,#72q ; Inițializează registrul A cu constanta $72_8 = 3A_{16} = 58_{10}$

4. O constantă binară, care trebuie să înceapă cu '#' și să se termine cu litera 'b' ('B').

AND #00111010b ; ȘI logic între acumulator și constanta $00111010_2 = 72_8 = 3A_{16}$.

5. Valoarea curentă a contorului de program (PC) sau adresa instrucțiunii curente, care se specifică prin caracterul '\$'.

LJMP \$+8 ; Salt în program peste 8 locații de memorie, la adresa curentă +8.

6. O constantă ASCII inclusă între ghilimele simple.

MOV A,'@' ; Încarcă în acumulator codul ASCII al caracterului @ (= 40h).

7. Numele unei constante simbolice căreia asamblorul i-a atribuit o valoare numerică.

val: EQU #40h
ADD A,val ; Adună la registrul A valoarea constantei "val", definite anterior (40h).

8. Numele unei adrese simbolice (etichete), definite în segmentul de cod sau de date.

MOV DPTR,contor
MOVX A,@DPTR ; Încarcă în acumulator octetul de la adresa "contor".
buc1a: DEC A ; Decrementează conținutul acumulatorului.
JNZ buc1a ; Dacă, după decrementare, $A \neq 0$, execută un salt la
contor: DB 10 ; Instrucțiunea de la adresa "buc1a".
 ; Rezervă un octet, la adresa "contor"

9. Numele unui registru SFR, adresa unei locații de memorie de RAM interne adresate direct sau o adresă de bit.

PUSH ACC ; depune pe stivă conținutul registrului A (văzut ca SFR, cu adresa $0E0h$)
POP DPH ; extrage din stivă un octet pe care îl încarcă în DPH
PUSH 00h ; Pune pe stivă conținutul registrului R0 (adresa de RAM internă $00h$)
POP 40h ; Extrage din stivă un octet, pe care îl încarcă în memoria RAM internă, la adresa $40h$
SETB ACC.0 ; Setează bitul cel mai puțin semnificativ al registrului acumulator.
MOV 21h,C ; Inscrie valoarea flag-ului C în bitul cu adresa $21h$.

10. *Expresii aritmetice și logice* în care se folosesc toate tipurile de date descrise mai sus și care constituie operanzii expresiei. Operanzii sunt conectați cu ajutorul operatorilor aritmetici: +, -, *, /, MOD (modulo), logici: NOT, AND, OR, XOR, LOW, HIGH, SHL, SHR (deplasare la stânga sau la dreapta), precum și cu ajutorul parantezelor (stânga și dreapta). Lungimea operanzilor luați în considerație la evaluarea expresiilor de către asamblor este de 16 biți.

Operatorii aritmetici realizează, în ordinea enumerării lor, *adunarea, scăderea, înmulțirea, împărțirea* întreagă, respectiv calculul *restului împărțirii* dintre 2 operanzi. Operatorii logici acționează la nivel de bit și produc *selecția, complementarea, produsul logic, suma logică* și respectiv *suma modulo 2* a argumentelor. Operatorii SHL și SHR produc deplasarea liniară a primului operand spre stânga, respectiv spre dreapta, cu un număr de poziții egal cu valoarea celui de-al doilea operand. În partea opusă deplasării se introduc un număr de zerouri egal cu numărul de deplasări.

Ordinea în care sunt executate operațiile dintr-o expresie este următoarea:

1. expresiile dintre paranteze,
2. LOW, HIGH,
2. *, /, MOD, SHL, SHR,
3. +, -,
4. NOT,
5. AND,
6. OR, XOR.

Operatorii MOD, SHL, SHR, NOT, AND, OR și XOR trebuie separați de operanzi cu cel puțin un blank.

Comentariul este alcătuit dintr-un șir de caractere alfanumerice și este utilizat de programator pentru descrierea operației executate de respectiva instrucțiune. Acest câmp este opțional. Câmpul comentariu trebuie precedat de un caracter ';', care îl separă de câmpul operand. De regulă acesta este precedat, pentru aliniere, de mai multe blankuri sau tabulatori (' ', TAB). Un comentariu se poate întinde pe mai multe rânduri, dar fiecare rând trebuie să înceapă cu un delimitator ';'.

6.5.3. Tehnici de programare specifice, pe grupe de instrucțiuni

În continuare, pentru diferite *grupe funcționale*, va fi descris pe scurt modul de utilizare al instrucțiunilor, cu evidențierea prin exemple a unor caracteristici specifice familiei MCS-51.

Instrucțiunile aritmetice pot fi definite mai compact prin tipul mnemonice, ca în tab.6.23, în care sunt evidențiate modurile de adresare și timpul de execuție la $f_{osc} = 12\text{MHz}$.

Tab.6.23

Mnemonica	Operația	Moduri adresare				Timp de execuție (us)
		Directă	Indirectă	La registru	Imediată	
ADD A,byte	$A = A + \text{byte}$	x	x	x	x	1
ADDC A,byte	$A = A + \text{byte} + C$	x	x	x	x	1
SUBB A,byte	$A = A - \text{byte} - C$	x	x	x	x	1
INC A	$A = A + 1$	Implicită (numai A)				1
INC byte	$\text{byte} = \text{byte} + 1$	x	x	x		1
INC DPTR	$\text{DPTR} = \text{DPTR} + 1$	Implicită (numai DPTR)				2
DEC A	$A = A - 1$	Implicită (numai A)				1
DEC byte	$\text{byte} = \text{byte} - 1$	x	x	x		1
MUL AB	$B; A = B \times A$	Implicită (numai A și B)				4
DIV AB	$A = \text{Int}[A/B],$ $B = \text{Mod}[A/B]$	Implicită (numai A și B)				4
DA A	Corecție zecimală	Implicită (numai A)				1

Astfel, instrucțiunea ADD A,byte, care adună variabila byte la acumulator, poate fi scrisă explicit astfel:

ADD A,5Eh	: adresare directă
ADD A,@R1	: adresare indirectă

ADD	A,R7	; adresare cu registru
ADD	A,#100	; adresare imediată

Rezultatul operației se păstrează în acumulator și indicatorii de condiție sunt afectați astfel: C și AC se setează dacă există un transport de la bitul 7 și respectiv bitul 3, iar OV este setat dacă există un transport de la bitul 6 la bitul 7, dar nu și de la bitul 7 la C. De asemenea, C indică o depășire la adunarea întregilor fără semn, iar OV indică o depășire la adunarea întregilor cu semn (un rezultat negativ la adunarea a două numere întregi pozitive sau un rezultat pozitiv de la doi operanzi negativi).

Pentru a evidenția mai bine flexibilitatea programării indusă de modurile de adresare, se vor exemplifica prin instrucțiunea ADD A,byte diferite posibilități de procesare în limbaj de asamblare:

a) Operarea asupra spațiului intern de memorie de date (primele 128 de locații RAM) și a zonei SFR prin adresare directă. În acest caz, instrucțiunea are forma particulară ADD A,direct; prin *direct* se specifică adresa de 1 octet a unei locații RAM interne.

Exemplu: Adunarea valorilor din două locații RAM adresate direct:

; Sumarea conținutului locației RAM cu adresa 31h la conținutul locației cu adresa 30h		
MOV	A,30h	; Se încarcă în acumulator conținutul locației 30h
ADD	A,31h	; Se adună cu conținutul locației 31h
MOV	30h,A	; Rezultatul se depune în locația 30h

Utilizând directive pentru alocarea statică a variabilelor (DB sau DS), exemplul de mai sus se poate scrie simbolic astfel:

var0:	ISEG	AT	30h	; Început de segment absolut de date, în memoria RAM internă.
var0:	DS	1		; Rezervă un octet la adresa var0 (30h).
var1:	DS	1		; Rezervă un octet la adresa var1 (31h).
;				
COD	SEGMENT	CODE		; COD este un segment relocabil de tip CODE.
	RSEG	COD		; Segmentul relocabil COD
;				
	MOV	A,var0		; Încarcă în A valoarea variabilei var0.
	ADD	A,var1		; Adună valoarea variabilei var1.
	MOV	var0,A		; Memorează rezultatul în variabila var0.

Exemplu: Adunarea datei unui port de intrare la data unui port de ieșire

; Adună octetul citit din portul P1 la data existentă la ieșirea portului P3.		
MOV	A,P0	
ADD	A,P1	
MOV	P0,A	

b) Operarea asupra memoriei RAM (interne sau externe) prin adresare indirectă. Forma particulară a instrucțiunii este ADD A,@Ri, unde @Ri specifică unul din registrele R0 sau R1 ale bancului curent, care conțin adresa locației RAM.

Exemplu: Adunarea conținutului a două locații din RAM adresate indirect.

; Adună conținutul locației din RAM specificat cu R1 la conținutul locației din RAM specificată cu R0		
MOV	A,@R0	
ADD	A,@R1	
MOV	@R0,A	

Obs.: După resetarea microcontrolerului se selectează implicit bancul 0; selectarea unui alt banc, deci și a registrelor R0 și R1 corespunzătoare, necesită modificarea în acord a biților PSW.4 și PSW.3.

c) Operarea asupra conținutului registrelor R0+R7 ale bancului selectat cu PSW. Forma explicită a instrucțiunii este ADD A,Rn, unde prin Rn se specifică registrul folosit ($n = 0 \div 7$).

Exemplu: Sumarea a două registre de date din bancul selectat.

; Adună conținutul registrului R3 la conținutul registrului R2

```
MOV  A,R2
ADD  A,R3
MOV  R2,A
```

d) Operarea cu constante specificate prin adresare imediată. Instrucțiunea are în acest caz forma explicită `ADD A,#data`, unde prin `#data` se specifică constanta utilizată în operație.

Exemplu: Adunarea unei constante, folosind adresarea imediată.

; Adună constanta 12h cu conținutul registrului R0; rezultatul se obține în A

```
MOV  A,#12h
ADD  A,R0
```

Instrucțiunea `ADDC A,byte` este similară precedentei, dar la rezultatul adunării se adaugă și conținutul indicatorului de transport. De asemenea, operandul sursă *byte* poate fi adresat *direct*, *indirect*, *cu registru* și *imediat*.

Instrucțiunea `SUBB A,byte` realizează scăderea cu împrumut din acumulator a operandului *byte*, care poate fi specificat *direct*, *indirect*, *cu registru* sau *imediat*. Rezultatul se obține în A, iar indicatorii de condiție C, AC și OV sunt afectați în conformitate cu rezultatul operației, similar instrucțiunii precedente.

Datorită faptului că `ADDC` și `SUBB` iau în calcul starea precedentă a lui C, pot fi realizate calcule în precizie multiplă prin repetarea operației cu octeții succesivi ai operandului. În ambele cazuri, fanionul de transport trebuie resetat la începutul primei iterații. La operațiile la care datele de intrare sunt cu semn, în complement față de doi, va fi setat și fanionul OV dacă apare transport (`ADDC`) sau împrumut (`SUBB`). În ultimul caz, testarea indicatorului OV prin instrucțiuni pe bit (`JB` sau `JNB`), permite corectarea rezultatului în anomaliile care pot apare (valoarea rezultatului depășește posibilitatea de reprezentare cu șapte biți și un bit de semn).

Exemplu: Scăderea a două numere cu semn, cu precizie impusă și cu detectarea erorilor de depășire.

```
; SUBSTR - rutină de scădere a șirului indicat de R1 din șirul indicat de R0, cu lungimea specificată de
;                                registrul R2
substr: CLR  C                ; BORROW = 0
subs1: MOV  A,@R0
        SUBB A,@R1            ; Scăderea poziției următoare
        MOV  @R0,A
        INC  R0                ; Actualizare pointeri
        INC  R1
        DJNZ R2,subs1         ; Buclare în funcție de precizie
        JNB  OV,ov_ok         ; Test dacă apare depășire la ultima iterație
;                                (Rutină de corecție)
ov_ok:  RET                  ; Revenire în programul apelant.
```

Dintre instrucțiunile de incrementare/decrementare trebuie evidențiate cele de tipul `INC/DEC byte`, care permit și incrementarea/decrementarea oricărui octet din memoria de date fără trecerea lor prin acumulator. O altă instrucțiune remarcabilă este cea de incrementare pe 16 biți a registrului `DPTR`, care generează adresa de 16 biți pentru memoria externă.

Instrucțiunile `MUL AB` și `DIV AB` permit multiplicarea și respectiv împărțirea numerelor întregi fără semn în doar 4 μ s (la $f_{osc} = 12\text{MHz}$), ceea ce constituie un avantaj major în raport cu microprocesoarele de 8 biți de uz general. La operația de multiplicare a acumulatorului cu registrul B, octetul inferior al rezultatului de 16 biți se obține în A, iar octetul superior în B. Dacă produsul este mai mare de 255 (0FFh) este setat fanionul OV; în caz contrar (octetul superior al produsului este zero), fanionul OV este resetat. Programatorul poate testa acest indicator pentru a

determina când B este diferit de zero și trebuie procesat. Fanionul de transport rămâne tot timpul pe zero. La operația de împărțire, câtul se returnează în A iar restul în B. Dacă inițial conținutul registrului B este 00h, atunci OV va fi setat, indicând o eroare de împărțire, iar valorile returnate vor fi indefinite. Și în acest caz fanionul C=0. Instrucțiunea de împărțire poate fi utilizată avantajos și în alte situații, cum ar fi conversia de bază de numerație ("radix conversion") sau separarea câmpurilor de biți din acumulator în semi-octeți ("nibbles").

Exemplu: Conversia unui întreg fără semn de un octet din binar în BCD.

```

; BINBCD - Rutină de conversie a unei variabile binare de 8 biți din acumulator în forma BCD
;           împachetat pe 3 digiți. Rangul sutelor se plasează în variabila "sute" iar a zecilor și
;           unităților în "zec_un". Adresele acestor locații sunt fixe, fiind definite prin directive EQU.
sute      EQU    21h
zec_un    EQU    22h
;
binbcd:   MOV     B,#100          ; Împarte cu 100 pentru a determina numărul sutelor.
          DIV     AB
          MOV     SUTE,A
          MOV     A,#10          ; Împarte restul la 10, pentru a determina numărul zecilor.
          XCH     A,B
          DIV     AB             ; Zecile în A, restul este cifra unităților.
          SWAP    A
          ADD     A,B            ; Împachetează cifrele BCD în A
          MOV     ZEC_UN,A
          RET

```

Pentru separarea unei date de un octet, în BCD, în cei doi semi-octeți componenți se poate utiliza o împărțire prin 16 a datei, cu reținerea semiocetului superior în A și a celui inferior în B. Cele două cifre obținute pot fi folosite individual pentru operare. Spre exemplu, printr-o operație de multiplicare se poate obține produsul celor doi digiți conținuți în A, după o prealabilă separare prin divizarea cu 16.

Exemplu: Implementarea înmulțirii în BCD prin utilizarea operațiilor MUL și DIV.

```

; MULBCD - Rutină de înmulțire a 2 digiți neîmpachetați recepționați în A,
;           cu returnarea produsului sub forma BCD împachetată.
mulbcd:   MOV     B,#10h         ; Împarte prin 16 conținutul lui A.
          DIV     AB             ; A&B rețin digiții separați.
          MUL     AB             ; A reține produsul în format binar.
          MOV     B,#10         ; Împarte produsul prin 10.
          DIV     AB             ; A reține zecile iar B unitățile.
          SWAP    A
          ORL     A,B            ; Împachetează digiții.
          RET

```

Ca și în cazul microprocesoarelor de uz general, este posibilă adunarea zecimală prin utilizarea instrucțiunii DA A în conjuncție cu ADD și/sau ADDC.

Exemplu: Adunarea operanzilor zecimali de 2 octeți conținuți în registre pereche, cu constante.

```

; BCDADD - Adunarea constantei 2345 (în zecimal) cu conținutul : "registrului pereche" R3R2
;           care conțin o variabilă de 4 digiți în BCD.
bcdadd:   MOV     A,R2
          ADD     A,#45h         ; Sumare octet inferior.
          DA      A              ; Corecție zecimală.
          MOV     R2,A           ; Rezultatul depus în R2.
          MOV     A,R3
          ADDC    A,#23h         ; Sumare octet superior cu transport
          DA      A              ; Corecție zecimală.
          MOV     R3,A           ; Rezultatul depus în R3.
          RET

```


Instrucțiunile logice pot fi definite, după tipul mnemonicelor, ca în tab.6.24. Sunt evidențiate modurile posibile de adresare, precum și durata de execuție la $f_{osc} = 12\text{MHz}$.

Tab.6.24

Mnemonică	Operația	Moduri adresare				Timp de execuție (μs)
		Directă	Indirectă	La Registru	Imediată	
ANL A,byte	$A = A \wedge \text{byte}$	x	x	x	x	1
ANL byte, A	$\text{byte} = \text{byte} \wedge A$	x				1
ANL byte,#data	$\text{byte} = \text{byte} \wedge \text{data}$	x				2
ORL A,byte	$A = A \vee \text{byte}$	x	x	x	x	1
ORL byte,A	$\text{byte} = \text{byte} \vee A$	x				1
ORL byte,#data	$\text{byte} = \text{byte} \vee \text{data}$	x				2
XRL A,byte	$A = A \oplus \text{byte}$	x	x	x	x	1
XRL byte,A	$\text{byte} = \text{byte} \oplus A$	x				1
XRL byte,#data	$\text{byte} = \text{byte} \oplus \text{data}$	x				2
CLR A	$A = 00\text{h}$	Implicită (numai A)				1
CPL A	$A = \neg A$	Implicită (numai A)				1
RL A	Rotește A la stânga 1 bit	Implicită (numai A)				1
RLC A	Rotește A la stânga prin C	Implicită (numai A)				1
RR A	Rotește A la dreapta 1 bit	Implicită (numai A)				1
RRC A	Rotește A la dreapta prin C	Implicită (numai A)				1
SWAP A	Schimbă semiocteții în A	Implicită (numai A)				1

Instrucțiunile ANL, ORL și XRL realizează funcțiile logice AND, OR și respectiv XOR asupra variabilelor de un octet. Nu sunt afectați indicatorii de condiție. Aceste operații pot utiliza aceleași moduri de adresare ca și cele aritmetice, cu deosebirea că nu restricționează operarea cu acumulatorul. Astfel, octeții adresați direct pot fi utilizați ca destinație, având ca sursă fie acumulatorul, fie o constantă. Aceste instrucțiuni sunt utile pentru ștergerea (ANL), setarea (ORL) sau complementarea (XRL) unuia sau mai multor biți din RAM, porturile de ieșire sau registrele de comandă. De exemplu, instrucțiunea XRL byte,#data oferă o modalitate rapidă și simplă pentru inversarea biților la pinii unui port.

Exemplu: Inversarea biților portului x ($x = 0, 1, 2$ sau 3).

```
Inv_px: XRL    P1,0FFH    ; Inversarea biților la pinii portului P1.
```

Dacă operația reprezintă un răspuns la o întrerupere, neutilizarea acumulatorului salvează timp și efort de lucru cu stiva în rutina de tratare. Un alt exemplu este cel de configurare a unui port cu ajutorul operațiilor logice: mai întâi o ștergere a biților ce trebuie modificați, urmată de o setare a acestora.

Exemplu: Reconfigurarea dimensiunii unui port de ieșire pentru ultimii 5 biți.

```
out_px: ANL    P1,#11100000b    ; Ștergerea biților P1.4+P1.0.
        ORL    P1,A              ; Setarea pinilor corespunzători la valoarea biților din A.
        RET
```

În exemplul de mai sus, biții care urmează să treacă pe nivel ridicat pot "pulsa" ("glitch") la nivel coborât pentru un ciclu mașină, datorită operației ANL. Dacă acest fapt este indesezirabil, poate fi folosită o altă abordare: mai întâi se setează toți pinii în conformitate cu conținutul registrului A și apoi se șterg biții din A care trebuie modificați.

Exemplu: Reconfigurarea dimensiunii unui port cu eliminarea impulsurilor parazite.

```
alt_px: ORL    P1,A
        ORL    A,#11100000b
        ANL    P1,A
        RET
```


Deși nu toți biții se vor schimba simultan din starea inițială în cea finală, nu va mai apare nici o tranziție intermediară a acestora.

Instrucțiunile de rotire (RL A, RRA, RLC A, RRC A) deplasează conținutul acumulatorului cu 1 bit la stânga sau la dreapta, fără sau prin fanionul de transport. Instrucțiunea SWAP A realizează interschimbarea semi-octeților acumulatorului, ceea ce este util, după cum s-a văzut, la operarea în BCD.

Instrucțiunile de transfer pot fi subîmpărțite în trei grupe, și anume: pentru transfer în RAM-ul intern, în RAM-ul extern și din memoria program.

Instrucțiunile de transfer în RAM-ul intern sunt prezentate în formă generalizată în tab.6.25, împreună cu modurile de adresare aferente, la $f_{osc} = 12\text{MHz}$.

Tab.6.25

Mnemonica	Operația	Moduri de adresare				Timp de execuție (μs)
		Directă	Indirectă	La registru	Imediată	
MOV A,src	$A \leftarrow \text{src}$	x	x	x	x	1
MOV dest,A	$\text{dest} \leftarrow A$	x	x	x		1
MOV dest,src	$\text{dest} \leftarrow \text{src}$	x	x	x	x	2
MOV DPTR,#data16	$\text{DPTR} \leftarrow \text{data16}$				x	2
PUSH src	INC SP; MOV "@SP", src	x				2
POP dest	MOV dest, "@SP"; DEC SP	x				2
XCH A,byte	A și byte schimbă datele	x	x	x		1
XCHD A,@ Ri	A și @Ri schimbă semioceteții inferiori		x			1

Primele două tipuri de instrucțiuni realizează schimbul de informații dintre acumulator și orice altă resursă internă (memorie RAM internă, porturi sau registre). Instrucțiunea MOV A,src admite ca sursă și o constantă.

Instrucțiunea MOV dest,src permite transferul de date între două locații din RAM sau din zona SFR fără trecerea prin acumulator, ceea ce asigură o flexibilitate sporită sistemelor cu MCS-51.

De departe aceste tipuri de instrucțiuni sunt cele mai flexibile și permit 15 combinații de surse și moduri de adresare. Această flexibilitate poate fi evidențiată și prin următorul

Exemplu: Locația din RAM cu adresa 30h conține valoarea 40h, iar locația 40h conține 10h. Data prezentă la intrarea portului 1 este 11001010b (0CAh), iar bancul de lucru curent este bancul 0 de registre. Atunci au loc următoarele transferuri:

MOV R0,#30h	: $R0 \leftarrow 30H$
MOV A,@R0	: $A \leftarrow 40H$
MOV R1,A	: $R1 \leftarrow 40H$
MOV B,@R1	: $B \leftarrow 10H$
MOV @R1,P1	: $(40H) \leftarrow 0CAH$
MOV P2,P1	: $P2 \leftarrow 0CAH$
MOV PSW,#00010000b	: $RS1 \leftarrow 1, RS0 \leftarrow 0$

care aduc valoarea 30h în R0, valoarea 40h în A și R1, valoarea 10h în B și valoarea 11001010b în locația RAM cu adresa 40h și la ieșirea Portului 2. Se selectează, în final, bancul 2 de registre.

Instrucțiunile de transfer includ și o încărcare a registrului DPTR cu o adresă de 16 biți, fie pentru o căutare într-un tabel din memoria program, fie pentru un acces de 16 biți în memoria externă de date.

Exemplu: Instrucțiunea va încărca valoarea 7C2Eh în DPTR: DPH va reține 7Ch iar DPL va reține 2Eh.

MOV DPTR,#7C2Eh

Instrucțiunile de transfer nu afectează indicatorii de condiție.

Pentru *lucrul cu stiva*, care se organizează în RAM-ul intern și crește "în sus", instrucțiunile **PUSH src** și **POP dest** asigură, prin adresare directă, salvarea/refacerea resurselor utilizate sau a căror conținut se alterează la lucrul cu subrutine. Stiva propriu-zisă se accesează indirect, prin intermediul registrului SP. Acest fapt permite plasarea stivei oriunde în spațiul RAM intern disponibil, inclusiv în cei 128 octeți superiori, dacă aceștia există. O situație tipică este cea a rutinelor de servire a întreruperilor, în care prin operații **PUSH-POP** se salvează și se refac prin stivă resursele ce pot fi distruse, spre exemplu la utilizarea lor în selectarea unui alt banc.

Exemplu: Utilizarea stivei pentru salvarea stării programului la o întrerupere pe linia $\overline{\text{INT0}}$.

CSEG	AT	0003h	
LJMP	rutint0		;Salt la rutina de servire dispusă oriunde în spațiul de memorie
rutint0:	PUSH	PSW	;Salvează registru PSW
	PUSH	ACC	;Salvează acumulatorul
	PUSH	B	;Salvează registrul B
	PUSH	DPL	;Salvează registrul DPTR
	PUSH	DPH	
	MOV	PSW,#00001000b	; Selectează bancul 1 de registre
			; (Corpul rutinei)
	POP	DPH	; Refacere registre în ordine inversă
	POP	DPL	
	POP	B	
	POP	ACC	
	POP	PSW	; Refacere PSW și revenire la bancul original de registre.
	RETI		; Revenire din întrerupere.

Dacă în momentul detectării întreruperii registrul SP conține adresa 2Eh, atunci în timpul execuției rutinei de servire stiva va avea configurația din fig.6.46. Registrul SP va conține 36h. Exemplul considerat reprezintă situația cea mai generală. Dacă, spre exemplu, rutina de servire nu va afecta registrele B și DPTR, atunci nu va fi necesară salvarea/refacerea acestora.

Instrucțiunile **PUSH-POP** pot fi utilizate avantajos, ca și la microprocesoarele de uz general, pentru transmiterea de variabile la / de la subrutine prin intermediul stivei.

Instrucțiunea **XCH A,byte** cauzează un schimb de date între acumulator și octetul adresat. Instrucțiunea **XCHD A,@Ri** permite schimbul semiocteților inferiori ai acumulatorului și octetului din RAM indirect adresat cu R0 sau R1. Aceste instrucțiuni permit schimbări rapide de conținut între acumulator și diferite locații din RAM sau din zona SFR. De asemenea, sunt avantajoase la manipulări de date sub formă zecimală.

Exemplu: Deplasarea la dreapta cu două cifre a unui număr în BCD de 8 digiți dispus în RAM-ul intern la 4 adrese succesive: $\text{adr1}+\text{adr4}$. Locația eliberată va avea conținut zero iar cifrele pierdute prin deplasare rămân în acumulator.

CLR	A	; A = 00h
XCH	A,adr1	; (adr1) ← 0, (A) ← (adr1)
XCH	A,adr2	; (adr2) = (adr1), (A) ← (adr2)
XCH	A,adr3	; (adr3) = (adr2), (A) ← (adr3)
XCH	A,adr4	; (adr4) = (adr3), (A) ← (adr4)

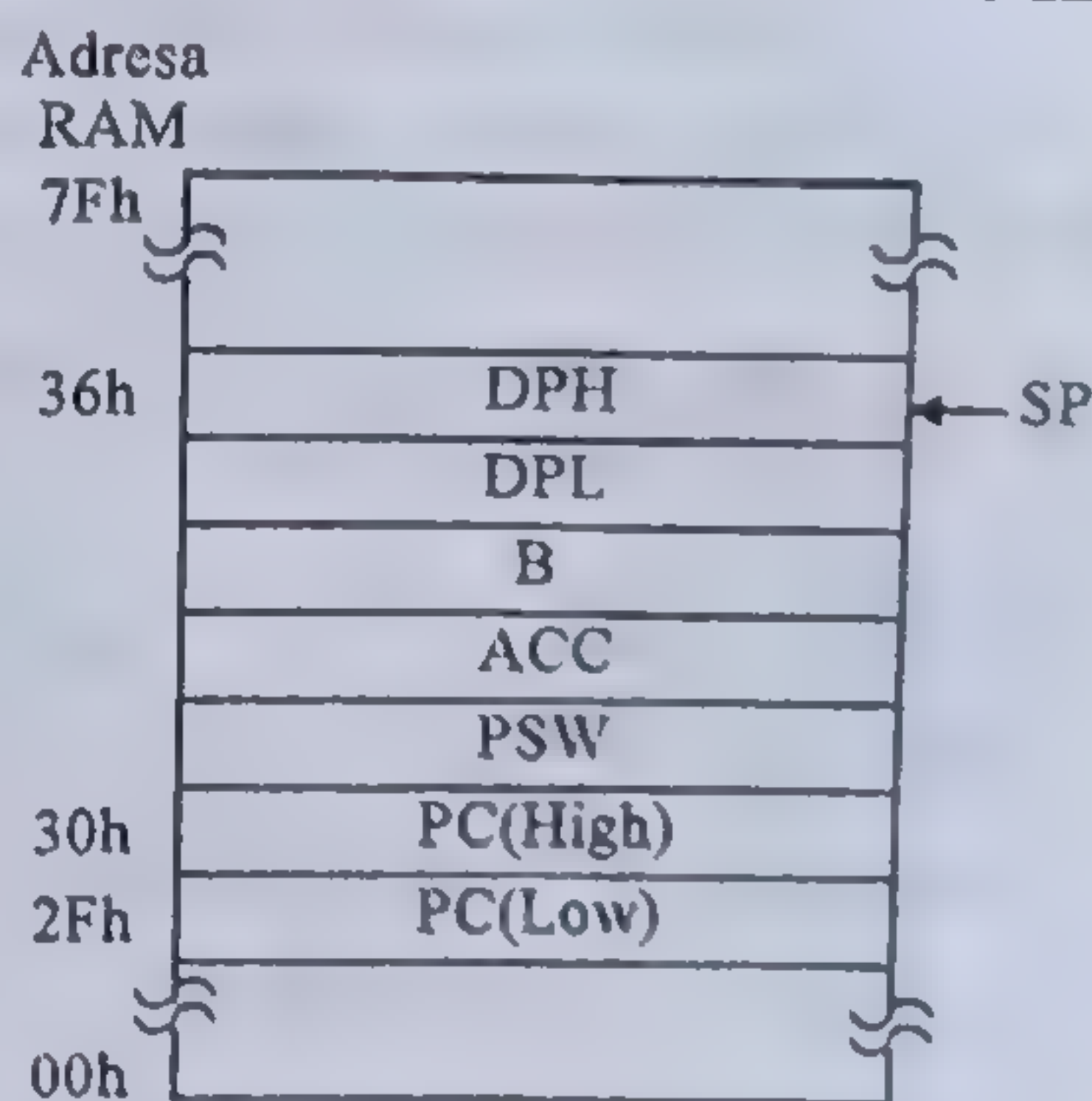


Fig.6.46. Conținutul stivei pe durata întreruperii

O aceeași operație de deplasare, realizată clasic cu instrucțiuni MOV A,src și MOV dest,src, durează aproape dublu (9 μ s față de 5 μ s, la $f_{osc} = 12\text{MHz}$).

La deplasarea cu un singur digit în BCD este avantajos să se folosească și instrucțiunea XCHD A, @Ri.

Instrucțiunile de transfer în RAM-ul extern folosesc adrese pe 8 sau 16 biți și sunt prezentate în tab.6.26. Pentru aceste instrucțiuni poate fi utilizată numai adresarea indirectă.

Tab.6.26

Dimensiune adresă	Mnemonica	Operația	Timp de execuție (μ s)
8 biți	MOVX A,@Ri	Citește RAM extern adresat indirect cu Ri	2
8 biți	MOVX @Ri,A	Scrie în RAM extern adresat indirect cu Ri	2
16 biți	MOVX A,@DPTR	Citește RAM extern adresat indirect cu DPTR	2
16 biți	MOVX @DPTR,A	Scrie în RAM extern adresat indirect cu DPTR	2

În cazul extensiei RAM până la 256 octeți este avantajos să se utilizeze primele două tipuri de instrucțiuni din tab.6.26, cu folosirea Portului 0 pentru generarea adresei în exterior. În astfel de situații nu este avantajoasă folosirea adreselor de 16 biți datorită utilizării incomplete a Portului 2, care transmite octetul superior din DPTR. La memorii de mici dimensiuni, dar mai mari de 256 octeți, mai avantajoasă este soluția cu paginarea memoriei RAM. Evident, în cazul unei dimensiuni mari a memoriei externe, cea mai eficace este adresarea pe 16 biți prin DPTR.

Trebuie remarcat faptul că în toate situațiile de acces la memoria RAM externă, acumulatorul este fie registrul sursă, fie cel destinație. Se reamintește că la execuția acestor instrucțiuni sunt generate automat în exterior semnalele \overline{RD} sau \overline{WR} .

Instrucțiunile de transfer din memoria program sunt prezentate în tab.6.27 și permit numai citirea datelor dispuse în memoria ROM/EPROM a sistemului. În cazul în care accesul se

Tab.6.27.

Mnemonica	Operația	Timp de execuție (μ s)
MOVC A,@+DPTR	Citește memoria program de la (A+DPTR)	2
MOVC A,@+PC	Citește memoria program de la (A+PC)	2

realizează la memoria program externă, atunci semnalul de citire este \overline{PSEN} . Ambele instrucțiuni utilizează o formă de adresare indexată: formatorul intern de adresă adună la adresa de bază de 16 biți, din DPTR sau PC, conținutul acumulatorului, care formează deplasamentul (indexul) de 8 biți fără semn. Rezultatul este transmis sub formă de adresă de 16 biți. Conținutul registrului DPTR nu este afectat, în schimb registrul PC este incrementat cu 1 (v.tab.6.21).

Acest mod de operare facilitează căutarea în tabele, de unde a și provenit denumirea dată în literatura de firmă acestor instrucțiuni: "look-up table read instructions".

Spre exemplu, prima instrucțiune MOVC din tab.6.27 este avantajoasă pentru citirea conținutului tabelelor cu până la 256 intrări, numerotate de la 0 la 255. Numărul de intrare dorit se încarcă în A, iar în DPTR se încarcă adresa de început a tabelului. Execuția instrucțiunii MOVC A,@A+DPTR va copia în acumulator conținutul tabelului de la intrarea specificată. Avantajul acestei instrucțiuni constă în faptul că tabelul poate fi plasat oriunde în memoria de cod ("global table look-up").

Cea de a doua instrucțiune MOVC operează în același mod cu precedenta, cu deosebire că adresa de bază este în registrul PC, ceea ce face ca tabelul accesat să fie local ("local table look-up"). Și în acest caz mai întâi se încarcă în A numărul intrării în tabel și apoi este apelată rutina ce conține tabelul.

Exemplu: Transferul în acumulator a unei valori dintr-un tabel cu patru elemente definite prin directive DB. Numerele de intrare (entry) pot fi între 0 și 3.

	MOV	A,entry	; Încarcă numărul intrării în tabel.
	CALL	table	; Apel rutină de citire din tabel.
table:	INC	A	
	MOV	A, @A+PC	
	RET		
	DB	11h	
	DB	20h	
	DB	0A3h	
	DB	46h	

Dacă în momentul apelării rutinei table în A se află, spre exemplu, valoarea 01h, rutina va returna 20h în acumulator. Instrucțiunea INC A înaintea instrucțiunii MOVC este necesară pentru a "depăși" instrucțiunea RET ce se află la începutul tabelului. Ca urmare, un astfel de tabel poate avea maximum 255 intrări. În general, în cazul în care un număr de octeți de cod separă instrucțiunea MOVC de tabel, același număr se va aduna la acumulator.

Instrucțiunile pentru manipularea variabilelor booleene formează setul de instrucțiuni al procesorului boolean și este reluat în tab.6.28, împreună cu timpul de execuție la 12 MHz. Se utilizează numai adresarea directă a tuturor biților manipulați de acest procesor, situați între adresele (de octet) 20h și 2Fh în RAM-ul intern (fig.6.47a) și biții registrelor cu adresare la nivel de bit din SFR (fig.6.47b). Cele două hărți din fig.6.47 conțin adresele biților direct adresabili de procesorul boolean.

Din tab.6.28 se observă că pentru toți acești biți există un meniu complet de instrucțiuni logice (NOT, OR, AND, SET, CLEAR), de transfer (MOV), dar și de instrucțiuni de salt condiționat, prin care se pot lua decizii la nivel de bit (JC, JNC, JB, JNB, JBC). În cazul ultimului grup de instrucțiuni este utilizată și adresarea relativă, adresa bitului fiind specificată printr-un octet cu semn care se adună în complement față de 2 la conținutul registrului PC. După cum s-a menționat la începutul capitolului, bitul de condiție C este utilizat de procesorul boolean ca acumulator, fapt ce se reflectă în sintaxa instrucțiunilor. De aici rezultă și simplitatea transferurilor la nivel de bit prin instrucțiunile de tipul MOV *dest_bit,src_bit*, în care totdeauna este utilizat C.

Tab.6.28.

Mnemonică	Operația	Timp de execuție (μs)
ANL C,bit	C = C AND bit	2
ANL C,/bit	C = C AND (NOT bit)	2
ORL C,bit	C = C OR bit	2
ORL C,/bit	C = C OR (NOT bit)	2
MOV C,bit	C = bit	1
MOV bit,C	bit = C	2
CLR C	C = 0	1
CLR bit	bit = 0	1
SETB C	C = 1	1
SETB bit	bit = 1	1
CPL C	C = NOT C	1
CPL bit	bit = NOT bit	1
JC rel	Salt dacă C = 1	2
JNC rel	Salt dacă C = 0	2
JB bit,rel	Salt dacă bit = 1	2
JNB bit,rel	Salt dacă bit = 0	2
JBC bit,rel	Salt dacă bit = 1; bit = 0	2

Adresă octet (msb)	(lsb)	Adresă octet (msb)	Adresă bit (lsb)	Simbol SFR
7Fh		FFh		
2Fh	7Fh 7Eh 7Dh 7Ch 7Bh 7Ah 79h 78h	F0h	7Fh 7Eh 7Dh 7Ch 7Bh 7Ah 79h 78h	B
2Eh	77h 76h 75h 74h 73h 72h 71h 70h	E0h	7Fh 7Eh 7Dh 7Ch 7Bh 7Ah 79h 78h	ACC
2Dh	6Fh 6Eh 6Dh 6Ch 6Bh 6Ah 69h 68h	D0h	7Fh 7Eh 7Dh 7Ch 7Bh 7Ah 79h 78h	PSW
2Ch	67h 66h 65h 64h 63h 62h 61h 60h	C8h	7Fh 7Eh 7Dh 7Ch 7Bh 7Ah 79h 78h	T2CON*
2Bh	5Fh 5Eh 5Dh 5Ch 5Bh 5Ah 59h 58h	B8h	7Fh 7Eh 7Dh 7Ch 7Bh 7Ah 79h 78h	IP
2Ah	57h 56h 55h 54h 53h 52h 51h 50h	B0h	7Fh 7Eh 7Dh 7Ch 7Bh 7Ah 79h 78h	P3
29h	4Fh 4Eh 4Dh 4Ch 4Bh 4Ah 49h 48h	A8h	7Fh 7Eh 7Dh 7Ch 7Bh 7Ah 79h 78h	IE
28h	47h 46h 45h 44h 43h 42h 41h 40h	A0h	7Fh 7Eh 7Dh 7Ch 7Bh 7Ah 79h 78h	P2
27h	3Fh 3Eh 3Dh 3Ch 3Bh 3Ah 39h 38h	98h	7Fh 7Eh 7Dh 7Ch 7Bh 7Ah 79h 78h	SCON
26h	37h 36h 35h 34h 33h 32h 31h 30h	90h	7Fh 7Eh 7Dh 7Ch 7Bh 7Ah 79h 78h	P1
25h	2Fh 2Eh 2Dh 2Ch 2Bh 2Ah 29h 28h	88h	7Fh 7Eh 7Dh 7Ch 7Bh 7Ah 79h 78h	TCON
24h	27h 26h 25h 24h 23h 22h 21h 20h	80h	7Fh 7Eh 7Dh 7Ch 7Bh 7Ah 79h 78h	P0
23h	1Fh 1Eh 1Dh 1Ch 1Bh 1Ah 19h 18h			
22h	17h 16h 15h 14h 13h 12h 11h 10h			
21h	0Fh 0Eh 0Dh 0Ch 0Bh 0Ah 09h 08h			
20h	07h 06h 05h 04h 03h 02h 01h 00h			
00h	Bancurile de registre 0, 1, 2 și 3			

a) adresele biților din RAM

b) adresele biților din zona SFR (* numai la 8052)

Fig.6.47. Locațiile adresabile pe bit ale MCS-51

Adresarea la nivel de bit a principalelor registre de comandă, dar mai ales a porturilor I/E, facilitează o comunicație eficientă cu exteriorul.

Exemplu: Se dorește ca pinul 2 al Portului 1, de ieșire, să aibă același nivel cu cel existent pe linia 3 a Portului 3, de intrare. Secvența de mai jos este edificatoare prin simplitate și eficiență.

```
MOV C,P3.3      ; Transferă în C starea pinului de intrare P3.3.
MOV P1.2,C      ; Aduce pinul P1.2 la nivelul corespunzător valorii din C.
```

Trebuie evidențiat faptul că instrucțiunile logice pe bit nu includ operația XRL (Exclusive OR), o astfel de operație putând fi ușor implementată prin program. Spre exemplu, suma modulo 2 a biților *bit1* și *bit2*, cu rezultatul în C: $C = bit1 \oplus bit2$, corespunde următoarei secvențe:

```
MOV C, bit1
JNB bit2, continua
CPL C
```

continua:

Similar pot fi implementate prin program funcții logice complexe, întâlnite în automatizări discrete.

Exemplu: Să se implementeze prin program funcția logică de 6 argumente:

$$F = [U \cdot (V + W)] + (X \cdot \bar{Y}) + \bar{Y}$$

Această funcție poate fi reprezentată și sub forma organigramei din fig.6.48. În acest caz, implementarea SW se poate obține folosind numai instrucțiuni de test:

```

test_V: JB     V,test_U
        JNB    W,test_X
test_U: JB     U,set_F
test_X: JNB    X,test_Z
        JNB    Y,set_F
test_Z: JNB    Z,set_F
clr_F:  CLR    F
        JMP    continua
set_F:  SETB   F
continua:
    
```

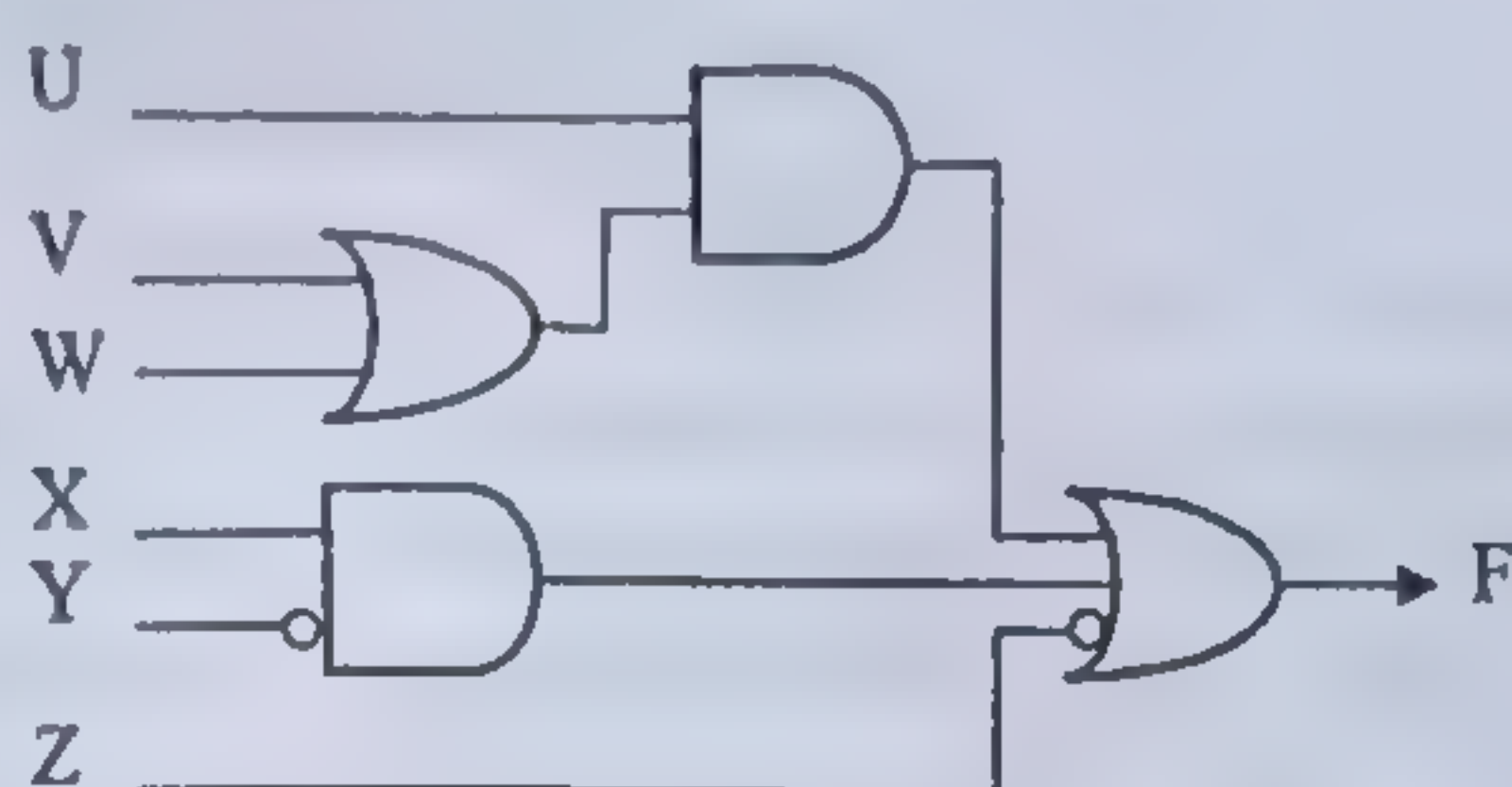


Fig.6.49. Schema logică a funcției F

O a doua rezolvare, mai elegantă, se poate obține plecând direct de la expresia logică sau de la schema echivalentă din fig.6.49.

```

MOV     C,V
ORL     C,W
ANL     C,U
MOV     F0,C ; Memorare rezultat intermediar.
MOV     C,X
ANL     C,/Y
ORL     C,F0 ; Ține cont de valoarea memorată.
ORL     C,/Z ; Include ultima variabilă.
MOV     F,C ; Evidențiază rezultatul.
    
```

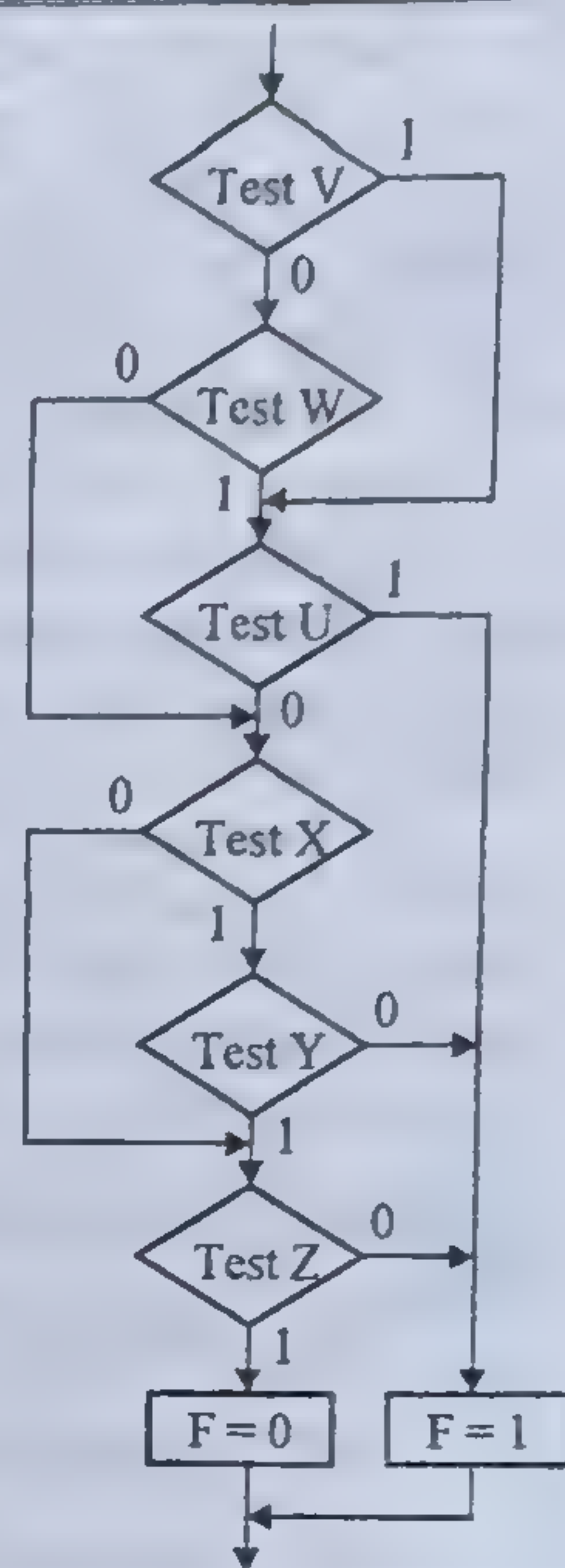


Fig.6.48. Organigrama funcției F

Instrucțiunile de ramificare sunt de tip necondiționat și de tip condiționat.

Instrucțiunile de salt necondiționat sunt reluate parțial generalizat în tab.6.29, împreună cu timpul de execuție la $f_{osc} = 12\text{MHz}$. Prima instrucțiune din tab.6.29 este una generică, cuprinzând în fapt trei instrucțiuni - SJMP, LJMP și AJMP - care diferă prin formatul adresei de salt (v. și tab.6.21). Astfel,

instrucțiunea SJMP *rel* (Short JuMP) codifică adresa destinație ca un deplasament de un octet cu semn care se adună la conținutul registrului PC, după incrementarea acestuia cu 2. Asamblorul interpretează corect forma SJMP *addr*, calculând automat deplasamentul cu relația $rel = addr - \$ - 2$, pe care îl amplasează în memorie imediat după opcodul instrucțiunii.

Exemplu: Salt "scurt" la adresa *addr*=0123h, efectuat cu o instrucțiune SJMP amplasată la adresa curentă $\$ = 0100h$. Deplasamentul calculat de asamblor va fi $rel = 123h - 100h - 2 = 21h$. La execuție, registrul PC se incrementează de două ori la extragerea celor doi octeți ai instrucțiunii, ajungând la valoarea 0102h. La aceasta se adună cel de-al doilea octet al instrucțiunii, $rel = 21h$, ceea ce are drept efect încărcarea în PC a valorii 0123h, adică tocmai adresa de salt dorită, *addr*.

Instrucțiunea LJMP *addr16* (Long JuMP) codifică adresa de salt printr-o constantă de 16 biți, permițând ramificarea oriunde în cei 64Ko ai spațiului de memorie program.

Tab.6.29

Mnemonică	Operația	Timp de execuție (us)
JMP <i>addr</i>	Salt la adresa <i>addr</i>	2
JMP @A+DPTR	Salt la adresa A+DPTR	2
CALL <i>addr</i>	Apel subrutină de la <i>addr</i>	2
RET	Revenire din subrutină	2
RETI	Revenire din întrerupere	2
NOP	Nici o operație	1

Instrucțiunea **AJMP** *addr11* codifică adresa de salt printr-o constantă de 11 biți, realizând ramificarea în interiorul unei pagini de memorie program de 2Ko, care urmează instrucțiunii (încărcarea registrului PC se face tot după incrementarea cu 2). Instrucțiunea are, ca și **SJMP**, numai 2 octeți (v.tab.6.21). Octetul de cod include și biții 8, 9 și 10 ai adresei, iar ultimul octet biții 7÷0. La momentul execuției, cei 11 biți ai noii adrese îi substituie pe cei existenți în PC, în timp ce biții 15÷11 rămân nealterați.

Și în acest caz programatorul va specifica adresa de salt în limbaj de asamblare (ASM-51) ca o etichetă sau ca o constantă de 16 biți. Asamblorul va pune adresa destinație în formatul corect pentru instrucțiunea dată. Dacă formatul cerut de instrucțiune nu va suporta distanța la adresa de salt specificată, atunci asamblorul va genera un mesaj corespunzător de eroare ("Destination out of range").

Instrucțiunea **JMP @A+DPTR** realizează un salt *indirect* la o adresă calculată la momentul execuției programului, ca suma dintre conținutul registrului DPTR și conținutul acumulatorului. Tipic, DPTR se încarcă cu adresa unui tabel de ramificații, iar A se încarcă cu un index (octet fără semn) ce reprezintă intrarea în tabel. Pot fi realizate facil cu ajutorul acestei instrucțiuni ramificări multiple în program, adresele de salt fiind organizate sub formă de tabel.

Exemplu: Ramificare multiplă în program, cu până la 128 de destinații posibile în aceeași pagină de 2Ko.

JNB	SCON.0,\$; Așteaptă recepția unui cod de comandă pe linia serială (SCON.0=RI).
MOV	A,SBUF	; Citește codul comenzii din tamponul de recepție serială.
RL	A	; Îl multiplică cu 2.
MOV	DPTR,#tabcom	; Adresa primei intrări în tabelul cu instrucțiuni de ramificare.
JMP	@A+DPTR	; Salt în tabelul de ramificări.
tabcom: ;128 de instrucțiuni AJMP succesive.		
AJMP	cmd0	; Salt la secvența de instrucțiuni pentru execuția comenzii cu codul 0.
AJMP	cmd1	; ...
AJMP	cmd2	
.....		
AJMP	cmd126	
AJMP	cmd127	

Multiplicarea indexului cu 2 este necesară deoarece intrările în tabel au câte doi octeți. Secvența de mai sus poate fi utilizată pentru execuția unor comenzi primite prin portul serial, a căror succesiune nu este dinainte cunoscută.

Instrucțiunea **CALL** *addr* din tab.6.29 generalizează două instrucțiuni similare - **LCALL** și **ACALL** - care diferă prin formatul adresei subrutinei apelate (v.tab.6.21). Astfel, instrucțiunea **LCALL** utilizează un format de adresă de 16 biți, permițând dispunerea subrutinelor oriunde în spațiul de 64 Ko al memoriei program. Cealaltă instrucțiune utilizează un format de 11 biți, ceea ce impune dispunerea subrutinelor în același bloc de 2 Ko cu instrucțiunea ce succede lui **ACALL**.

Codificarea adreselor rutinelor se realizează la fel ca în cazul instrucțiunilor corespunzătoare de salt. Subrutinele trebuie să se termine cu instrucțiunea **RET**, care întoarce execuția programului la instrucțiunea care succede instrucțiunii **CALL**. **RETI** are același efect ca și **RET** dar, în plus, logica de întrerupere în vederea acceptării unor întreruperi de nivel inferior sau de același nivel, aflate în așteptare. Începerea tratării cererii următoare se face după execuția a încă unei instrucțiuni de program. Dacă execuția instrucțiunii **RETI** nu corespunde încheierii unei subrutine de întrerupere, atunci funcțional aceasta este identică cu **RET**.

Trebuie evidențiat faptul că RETI nu reface automat registrul PSW și că nici alte registre, cu excepția PC, nu sunt afectate.

Instrucțiunile de salt condiționat sunt reluate în tab.6.30, împreună cu modurile de adresare și durata de execuție la $f_{osc} = 12\text{MHz}$. Toate aceste instrucțiuni specifică adresa de salt printr-o adresare relativă, deplasamentul fiind un octet cu semn, combinată cu una din adresările implicită, directă, indirectă, la registru sau imediată pentru condiția testată.

Tab.6.30

Mnemonica	Operația	Moduri adresare				Timp de execuție (μs)
		Directă	Indirectă	La registru	Imediată	
JZ rel	Salt dacă $A = 0$			Implicită (numai A)		2
JNZ rel	Salt dacă $A \neq 0$			Implicită (numai A)		2
DJNZ byte,rel	Decrementează și salt dacă byte $\neq 0$	x		x		2
CJNE A,byte,rel	Salt dacă $A \neq \text{byte}$	x			x	2
CJNE byte,#data,rel	Salt dacă byte $\neq \text{data}$		x	x		2

După cum s-a văzut la începutul capitolului, nu există în registrul PSW un indicator de condiție Z, pentru conținutul al acumulatorului. Instrucțiunile JZ rel și JNZ rel testează tocmai îndeplinirea sau nu a acestei condiții direct pe valoarea conținută în acel moment în acumulator.

Următoarele trei tipuri de instrucțiuni din tab.6.30 combină o operație la nivel de octet cu un salt condiționat bazat pe rezultat. Astfel, DJNZ decrementează registrul sau octetul adresat direct și efectuează un salt dacă rezultatul diferă de zero, fără a fi afectați indicatorii. Această instrucțiune este ideală pentru implementarea buclelor de program cu număr de pași cunoscut la intrare sau pentru adăugarea unor întârzieri moderate (între 2 și 512 cicluri mașină) cu o singură instrucțiune.

Spre exemplu, următoarea secvență:

MOV R2,#8
comuta:
CPL P1.7
DJNZ R2,#comuta

va bascula P1.7 de opt ori, generând pe linia 7 a portului 1 patru impulsuri cu durata de 3 μs (la $f_{osc} = 12\text{MHz}$). Similar, secvența:

CLR P1.6
MOV R3,#49
DJNZ R3,\$
SETB P1.6

introduce o buclă de întârziere de 99 μs la forțarea pe zero a liniei P1.6, cu numai două instrucțiuni. Se reamintește că simbolul "\$" semnifică "adresa instrucțiunii curente" în limbaj de asamblare ASM-51.

Instrucțiunile CJNE (Compare and Jump if Not Equal) compară octeții adresați implicit, direct, imediat, indirect sau cu registru și execută un salt la adresa relativă, dacă aceștia diferă. În cazul în care primul operand este mai mic decât al doilea, fanionul C este setat; în celelalte cazuri este resetat. Acest mod de operare face instrucțiunile CNJE avantajoase atât în controlul buclelor de program, cât și în realizarea comparațiilor de tipul "mai mare decât...", "mai mic decât..." ("greater than, less than")

Exemplu: Așteptare în buclă până când la liniile Portului 1 apare o anumită valoare:

CJNE A,P1,\$.

Instrucțiunea pentru identificarea momentului în care conținutul acumulatorului este identic cu al Portului 1, urmată și de trecerea pe zero a fanionului C.

Exemplu: Compararea conținutului registrului R7 cu o constantă dată.

CJNE	R7,#12h,diferit	; Dacă R7 ≠ 12h, salt la instrucțiunea de la adresa not_eq.
.....	Altfel (R7 = 12h), continuă ...
diferit: JC	mai_mic	; Dacă R7 < 12h ...
.....	Altfel (R7 > 12h), ...

Instrucțiunea CJNE poziționează indicatorul de condiție C, care apoi este testat la adresa diferită în vederea stabilirii mărimii lui R7 în raport cu constanta 12h ("mai mic sau mai mare")

6.5.4. Programarea perifericelor integrate

6.5.4.1. Programarea porturilor I/E

În paragraful precedent, prin exemplele considerate, s-a putut vedea modul în care se lucrează cu porturile I/E. Versatilitatea acestor resurse este remarcabilă și rezidă atât din structura de tip "quasi-bidirecțională", cât și din faptul că porturile au asociate registre latch adresabile la nivel de bit. Această structură pseudo-bidirecțională permite utilizarea aceluiași port atât ca intrare cât și ca ieșire, dacă pinii folosiți ca intrări au latch-urile de ieșire poziționate pe "1".

Instrucțiunile cu un singur operand (INC, DEC, DJNZ și CPL) acționează asupra conținutului latch-ului asociat, nu asupra pinului de intrare. Similar, la instrucțiunile cu doi operanzi care utilizează un port atât ca sursă cât și ca destinație (ANL, ORL, XRL) se folosesc ieșirile latch-urilor. Cel de-al doilea operand nu trebuie să altereze starea biților corespunzători pinilor de intrare. De asemenea, operația booleană JBC testează bitul latch-ului de ieșire și nu pinul de intrare în luarea deciziei de salt.

Un exemplu edificator de utilizare a facilităților menționate este cel de folosire a portului expander 8243 [49] al familiei MCS-48. După cum s-a văzut, MCS-51 nu conține instrucțiuni dedicate interfațării cu astfel de porturi. Conectarea unui 8243 ca port extern la MCS-51 implică un protocol emulat prin program, ca în exemplul următor.

Exemplu: Citirea datelor dintr-un expander 8243 prin intermediul Portului 2 (fig.6.50).

Datele se obțin prin 8243 conectat la P2.3+P2.0. Liniile P2.5 și P2.4 simulează

semnalele CS și PROG. Liniile P2.7 și P2.6 se utilizează ca intrări, iar datele se depun în acumulator.

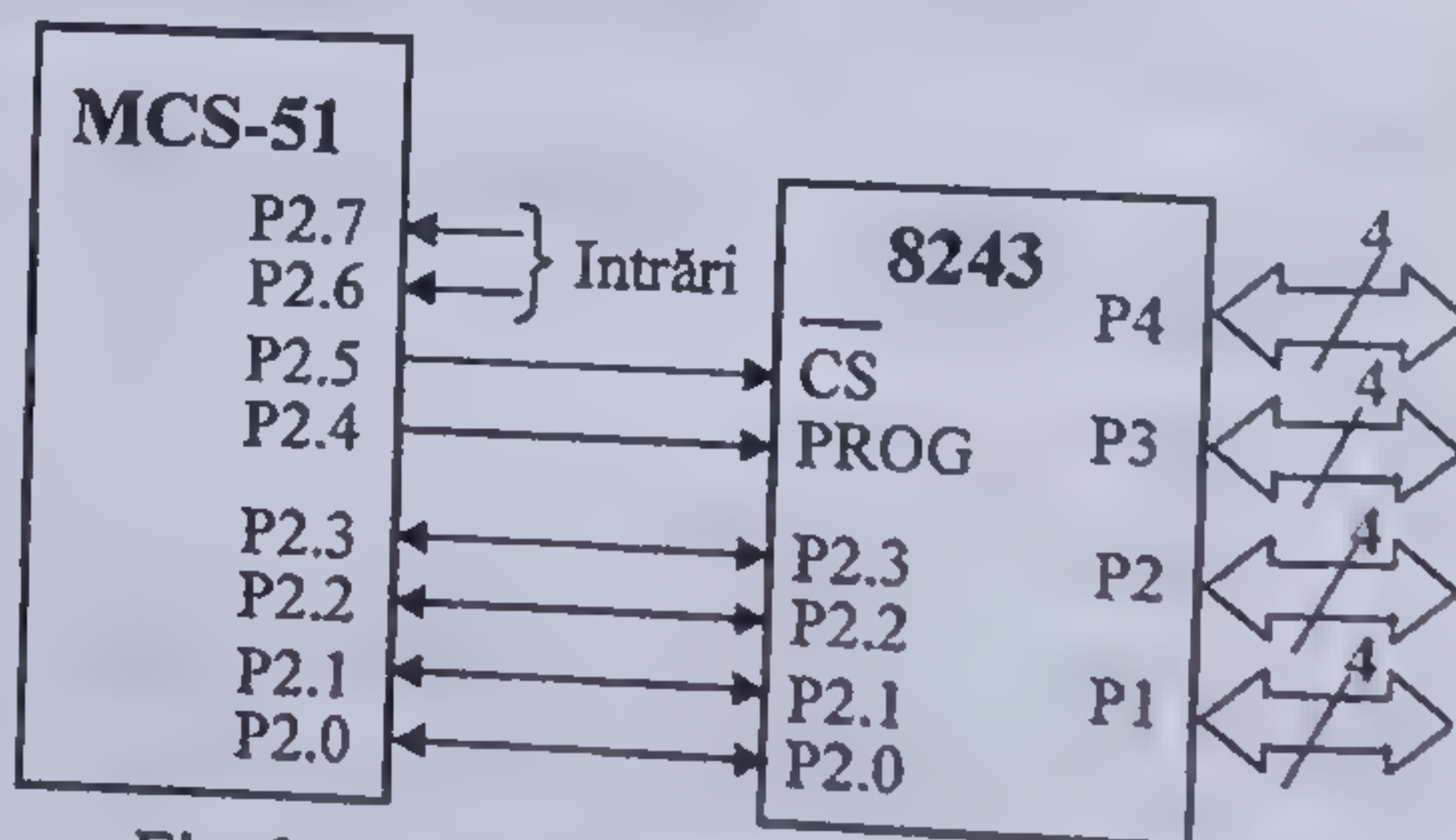


Fig.6.50. Conectarea expanderului 8243 la MCS-51

IN8243 - rutina de citire din expanderul 8253
 Intrare: A - codul Instrucțiunii de citire
 Ieșire: A - conține data citită
 Distrugă: nimic

in8243: ORL A,#11010000b
 MOV P2,A

CLR P2.4
 ORL P2,#00001111b
 MOV A,P2
 ANL A,#00001111b
 SETB P2.4
 SETB P2.5

; Depune cod instrucțiune pe pinii P2.5+P2.0, biții P2.6 și P2.7
 ; trebuie să rămână "1".
 ; Creează un front căzător pentru PROG.
 ; Setare pentru intrare.
 ; Citire date intrare.
 ; Reține doar biții citiți din expander.
 ; Revenire PROG pe "1".
 ; Deselectare 8243.

6.5.4.2. Programarea portului serial și a timerelor

Programarea portului serial se face în funcție de modul de lucru dorit. După conectarea sursei de alimentare sau după resetarea hardware este necesar să se înscrie cuvinte de control corespunzătoare în SFR-urile care gestionează atât portul serial cât și timerul utilizat ca generator de tact.

Exemplu: Inițializarea portului serial pentru comunicație în modul modul 1 (UART cu 8 biți și un bit de STOP).

În conformitate cu structura registrului de control al portului serial, SCON (v.fig.6.23), rezultă următoarea instrucțiune pentru inițializare:

```
; Inițializare port serial pentru modul UART cu 8 biți de date și un bit de STOP
MOV  SCON,#01010010b
```

în care s-a specificat modul (SM0, SM1 = 01) și activarea recepției (SM2 = 0, REN = 1).

Pentru efectuarea transferului serial este necesar să se stabilească viteza de comunicație și să se aleagă timerul care furnizează semnalul corespunzător de tact.

Exemplu: Comunicația serială din exemplul precedent se va realiza cu 2400 Baud și va fi utilizat timerul 1 în modul cu reîncărcare automată (modul 2).

Pentru a se asigura rata de transfer de 2400 Baud (la $f_{osc} = 12\text{MHz}$) este necesară încărcarea registrului TH1 cu valoarea rezultată din relația (v. §6.3.4.2):

$$V = \frac{2^{SMOD} \times f_{osc}}{32 \times 12[256 - (TH1)]}, [\text{Baud}],$$

cu SMOD = 0. După ce se fac înlocuirile și se efectuează calculele rezultă:

$$TH1 = 253_{10} = F3_{16}$$

Ținând cont de formatul registrelor TMOD (fig.6.16) și TCON (fig.6.17), secvența de inițializare a generatorului de tact pentru interfața serială este următoarea:

```
; Inițializare T1 în modul cu auto-încărcare, la 2400 Baud
;
MOV  TMOD,#00100000b  ; Stabilire mod de funcționare.
MOV  TH1, #0F3h        ; Încărcare constantă de timp.
SETB TR1               ; Pornire Timer 1 (prin TR1=1 în TCON)
```

Rutinele de transmisie și recepție pot fi construite să insereze și un bit de paritate, cu verificarea acestuia la primirea datelor.

Exemplu: Rutine pentru transmisie și recepție caracter, în modul UART cu 9 biți (8 biți de date și un bit de paritate), cu paritate pară.

```
; SER_OUT - Rutină de transmisie caracter de 8 biți, cu paritate pară
; Intrare: A - caracterul de transmis
; ieșire: Bitul de paritate al caracterului transmis se află în TB8
; Distrug: C
```

```
ser_out:  MOV  C,P          ; Pregătește bitul de paritate pentru a fi transmis.
          MOV  TB8,C        ; Așteaptă până când transmitătorul este disponibil.
          JNB  TI,$         ; Resetează TI, pregătindu-l pentru o semnalizare ulterioară.
          CLR  TI          ; Inscrie caracterul în tamponul de transmisie al portului serial.
          MOV  SBUF,A       ; Revine în programul apelant.
          RET
```


SER_IN - Rutină de recepție caracter de 8 biți, cu verificarea bitului de paritate (paritate pară)
Intrare: Nimic
Ieșire: A - caracterul recepționat.
Bitul de paritate al caracterului recepționat se află în RB8.
C = 1 dacă a apărut o eroare de paritate.
Distruge: A, C.

ser_in:	JNB	RI,\$; Așteaptă recepția unui caracter.
	CLR	RI	; Resetează RI, pregătindu-l pentru o semnalizare ulterioară.
	MOV	A,SBUF	; Citește caracterul din tamponul de recepție al portului serial.
	MOV	C,P	; Setează C la eroare de paritate, adică dacă RB8 ≠ P.
	JNB	RB8,revine	
	CPL	C	
revine:	RET		

h h ↓	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	LXI B,D16	STAX B	INX B	INR B	DCR B	MVI B,D8	RLC	-	DAD B	LDAX B	DCX B	INR C	DCR C	MVI C,D8	RRC
1	-	LXI D,D16	STAX D	INX D	INR D	DCR D	MVI D,D8	RAL	-	DAD D	LDAX D	DCX D	INR E	DCR E	MVI E,D8	RAR
2	RIM	LXI H,D16	SHLD addr	INX H	INR H	DCR H	MVI H,D8	DAA	-	DAD H	LHLD addr	DCX H	INR L	DCR L	MVI L,D8	CMA
3	SIM	LXI SP,D16	STA addr	INX SP	INR M	DCR M	MVI M,D8	STC	-	DAD SP	LDA addr	DCX SP	INR A	DCR A	MVI A,D8	CMC
4	MOV B,B	MOV B,C	MOV B,D	MOV B,E	MOV B,H	MOV B,L	MOV B,M	MOV B,A	MOV C,B	MOV C,C	MOV C,D	MOV C,E	MOV C,H	MOV C,L	MOV C,M	MOV C,A
5	MOV D,B	MOV D,C	MOV D,D	MOV D,E	MOV D,H	MOV D,L	MOV D,M	MOV D,A	MOV E,B	MOV E,C	MOV E,D	MOV E,E	MOV E,H	MOV E,L	MOV E,M	MOV E,A
6	MOV H,B	MOV H,C	MOV H,D	MOV H,E	MOV H,H	MOV H,L	MOV H,M	MOV H,A	MOV L,B	MOV L,C	MOV L,D	MOV L,E	MOV L,H	MOV L,L	MOV L,M	MOV L,A
7	MOV M,B	MOV M,C	MOV M,D	MOV M,E	MOV M,H	MOV M,L	HLT	MOV M,A	MOV A,B	MOV A,C	MOV A,D	MOV A,E	MOV A,H	MOV A,L	MOV A,M	MOV A,A
8	ADD B	ADD C	ADD D	ADD E	ADD H	ADD L	ADD M	ADD A	ADC B	ADC C	ADC D	ADC E	ADC H	ADC L	ADC M	ADC A
9	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB M	SUB A	SBB B	SBB C	SBB D	SBB E	SBB H	SBB L	SBB M	SBB A
A	ANA B	ANA C	ANA D	ANA E	ANA H	ANA L	ANA M	ANA A	XRA B	XRA C	XRA D	XRA E	XRA H	XRA L	XRA M	XRA A
B	ORA B	ORA C	ORA D	ORA E	ORA H	ORA L	ORA M	ORA A	CMP B	CMP C	CMP D	CMP E	CMP H	CMP L	CMP M	CMP A
C	RNZ	POP B	JNZ addr	JMP addr	CNZ addr	PUSH B	ADI D8	RST 0	RZ	RET	JZ addr	-	CZ addr	CALL addr	ACI D8	RST 1
D	RNC	POP D	JNC addr	OUT D8	CNC addr	PUSH D	SUI D8	RST 2	RC	-	JC addr	IN D8	CC addr	-	SBI D8	RST 3
E	RPO	POP H	JPO addr	XTHL	CPO addr	PUSH H	ANI D8	RST 4	RPE	PCHL	JPE addr	XCHG	CPE addr	-	XRI D8	RST 5
F	RP	POP PSW	JP addr	DI	CP addr	PUSH PSW	ORI D8	RST 6	RM	SPHL	JM addr	EI	CM addr	-	CPI D8	RST 7

Obs.: 1) Prin D8 și D16 s-au notat datele de unu sau doi octeți, iar prin addr o adresă de 16 biți.

2) Instrucțiuni specifice microprocesorului 8085.

Anexa II - Codurile instrucțiunilor microprocesorului Z80 - setul principal

h ↓	h ↑	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	LD BC,nn	LD (BC),A	INC B	INC BC	INC B	DEC B	LD B,n	RLCA	EX AF,AF'	ADD HL,BC	LD A,(BC)	DEC BC	INC C	DEC C	LD C,n	PPCA
1	DJNZ e	LD DE,nn	LD (DE),A	INC D	INC DE	INC D	DEC D	LD D,n	RLA	JR e	ADD HL,DE	LD A,(DE)	DEC DE	INC E	DEC E	LD E,n	PPA
2	JR e	LD HL,nn	LD (nn),L	INC H	INC HL	INC H	DEC H	LD H,n	DAA	JR Z,e	ADD HL,HL	LD HL,(nn)	DEC HL	INC L	DEC L	LD L,n	CPL
3	JR NC,e	LD SP,nn	LD (nn),A	INC (HL)	INC SP	INC (HL)	DEC (HL)	LD (HL),n	SCF	JR C,e	ADD HL,SP	LD A,(nn)	DEC SP	INC A	DEC A	LD A,n	CCF
4	LD B,B	LD B,C	LD B,D	LD B,H	LD B,E	LD B,H	LD B,L	LD B,(HL)	LD B,A	LD C,B	LD C,C	LD C,D	LD C,E	LD C,H	LD C,L	LD C,(HL)	CA
5	LD D,B	LD D,C	LD D,D	LD D,H	LD D,E	LD D,H	LD D,L	LD D,(HL)	LD D,A	LD E,B	LD E,C	LD E,D	LD E,E	LD E,H	LD E,L	LD E,(HL)	EA
6	LD H,B	LD H,C	LD H,D	LD H,H	LD H,E	LD H,H	LD H,L	LD H,(HL)	LD H,A	LD L,B	LD L,C	LD L,D	LD L,E	LD L,H	LD L,L	LD L,(HL)	LA
7	LD (HL),B	LD (HL),C	LD (HL),D	LD (HL),E	LD (HL),H	LD (HL),H	LD (HL),L	HALT	LD (HL),A	LD A,B	LD A,C	LD A,D	LD A,E	LD A,H	LD A,L	LD A,(HL)	AA
8	ADD A,B	ADD A,C	ADD A,D	ADD A,H	ADD A,E	ADD A,H	ADD A,L	ADD A,(HL)	ADD A,A	ADC A,B	ADC A,C	ADC A,D	ADC A,E	ADC A,H	ADC A,L	ADC A,(HL)	ADC
9	SUB B	SUB C	SUB D	SUB H	SUB E	SUB H	SUB L	SUB (HL)	SUB A	SBC A,B	SBC A,C	SBC A,D	SBC A,E	SBC A,H	SBC A,L	SBC A,(HL)	SBC
A	AND B	AND C	AND D	AND H	AND E	AND H	AND L	AND (HL)	AND A	XOR B	XOR C	XOR D	XOR E	XOR H	XOR L	XOR (HL)	XOR A
B	OR B	OR C	OR D	OR H	OR E	OR H	OR L	OR (HL)	OR A	CP B	CP C	CP D	CP E	CP H	CP L	CP (HL)	CP
C	RET NZ	POP BC	JP NZ,nn	JP	JP nn	CALL NZ,nn	PUSH BC	ADD A,n	RST 00h	RET Z	RET	JP Z,nn	Prefix	CALL Z,nn	CALL nn	ADC A,n	RST 08h
D	RET NC	POP DE	JP NC,nn	JP (n),A	OUT (n),A	CALL NC,nn	PUSH DE	SUB n	RST 10h	RET C	EXX	JP C,nn	IN A,(n)	CALL C,nn	Prefix IX	SBC A,n	RST 18h
E	RET PO	POP HL	JP PO,nn	EX (SP),HL	EX (SP),HL	CALL PO,nn	PUSH HL	AND n	RST 20h	RET PE	JP (HL)	JP PE,nn	EX DE,HL	CALL PE,nn	Prefix	XOR n	RST 28h
F	RET P	POP AF	JP P,nn	DI	DI	CALL P,nn	PUSH AF	OR n	RST 30h	RET M	LD SP,HL	JP M,nn	EI	CALL M,nn	Prefix IV	CP n	RST 38h

Instrucțiunile marcate permit drept operand și registrele IX sau IY, iar adresarea indirectă cu (HL) devine adresare indexată (IX+q), respectiv (IY+d). În acest caz, codul din tabel trebuie precedat de prefixul DDh, respectiv FEh. Un caz particular îl constituie instrucțiunile cu mnemonica ADD IX,pp (pp=BC, DE, IX sau SP), pentru care la codul din tabel se adună 40h.

h ↓	h ↓	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	RLC	RLC B	RLC C	RLC D	RLC E	RLC H	RLC L	RLC (HL)	RLC A	RRC B	RRC C	RRC D	RRC E	RRC H	RRC L	RRC (HL)	RRC A
1	RL	RL B	RL C	RL D	RL E	RL H	RL L	RL (HL)	RL A	RR B	RR C	RR D	RR E	RR H	RR L	RR (HL)	RR A
2	SLA	SLA B	SLA C	SLA D	SLA E	SLA H	SLA L	SLA (HL)	SLA A	SRA B	SRA C	SRA D	SRA E	SRA H	SRA L	SRA (HL)	SRA A
3	-	-	-	-	-	-	-	-	-	SRL B	SRL C	SRL D	SRL E	SRL H	SRL L	SRL (HL)	SRLA
4	BIT	BIT 0,B	BIT 0,C	BIT 0,D	BIT 0,E	BIT 0,H	BIT 0,L	BIT 0,(HL)	BIT 0,A	BIT 1,B	BIT 1,C	BIT 1,D	BIT 1,E	BIT 1,H	BIT 1,L	BIT 1,(HL)	BIT 1,A
5	BIT	BIT 2,B	BIT 2,C	BIT 2,D	BIT 2,E	BIT 2,H	BIT 2,L	BIT 2,(HL)	BIT 2,A	BIT 3,B	BIT 3,C	BIT 3,D	BIT 3,E	BIT 3,H	BIT 3,L	BIT 3,(HL)	BIT 3,A
6	BIT	BIT 4,B	BIT 4,C	BIT 4,D	BIT 4,E	BIT 4,H	BIT 4,L	BIT 4,(HL)	BIT 4,A	BIT 5,B	BIT 5,C	BIT 5,D	BIT 5,E	BIT 5,H	BIT 5,L	BIT 5,(HL)	BIT 5,A
7	BIT	BIT 6,B	BIT 6,C	BIT 6,D	BIT 6,E	BIT 6,H	BIT 6,L	BIT 6,(HL)	BIT 6,A	BIT 7,B	BIT 7,C	BIT 7,D	BIT 7,E	BIT 7,H	BIT 7,L	BIT 7,(HL)	BIT 7,A
8	RES	RES 0,B	RES 0,C	RES 0,D	RES 0,E	RES 0,H	RES 0,L	RES 0,(HL)	RES 0,A	RES 1,B	RES 1,C	RES 1,D	RES 1,E	RES 1,H	RES 1,L	RES 1,(HL)	RES 1,A
9	RES	RES 2,B	RES 2,C	RES 2,D	RES 2,E	RES 2,H	RES 2,L	RES 2,(HL)	RES 2,A	RES 3,B	RES 3,C	RES 3,D	RES 3,E	RES 3,H	RES 3,L	RES 3,(HL)	RES 3,A
A	RES	RES 4,B	RES 4,C	RES 4,D	RES 4,E	RES 4,H	RES 4,L	RES 4,(HL)	RES 4,A	RES 5,B	RES 5,C	RES 5,D	RES 5,E	RES 5,H	RES 5,L	RES 5,(HL)	RES 5,A
B	RES	RES 6,B	RES 6,C	RES 6,D	RES 6,E	RES 6,H	RES 6,L	RES 6,(HL)	RES 6,A	RES 7,B	RES 7,C	RES 7,D	RES 7,E	RES 7,H	RES 7,L	RES 7,(HL)	RES 7,A
C	SET	SET 0,B	SET 0,C	SET 0,D	SET 0,E	SET 0,H	SET 0,L	SET 0,(HL)	SET 0,A	SET 1,B	SET 1,C	SET 1,D	SET 1,E	SET 1,H	SET 1,L	SET 1,(HL)	SET 1,A
D	SET	SET 2,B	SET 2,C	SET 2,D	SET 2,E	SET 2,H	SET 2,L	SET 2,(HL)	SET 2,A	SET 3,B	SET 3,C	SET 3,D	SET 3,E	SET 3,H	SET 3,L	SET 3,(HL)	SET 3,A
E	SET	SET 4,B	SET 4,C	SET 4,D	SET 4,E	SET 4,H	SET 4,L	SET 4,(HL)	SET 4,A	SET 5,B	SET 5,C	SET 5,D	SET 5,E	SET 5,H	SET 5,L	SET 5,(HL)	SET 5,A
F	SET	SET 6,B	SET 6,C	SET 6,D	SET 6,E	SET 6,H	SET 6,L	SET 6,(HL)	SET 6,A	SET 7,B	SET 7,C	SET 7,D	SET 7,E	SET 7,H	SET 7,L	SET 7,(HL)	SET 7,A

Instrucțiunile marcate suportă și adresarea indexată, caz în care au prefixul DDh sau FDh.
În mnemonice se va folosi, în loc de (HL), (IX + d) și respectiv (IY + d).

h ↓ ↑	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	AJMP addr11	LJMP addr16	RR A	INC A	INC direct	INC @R0	INC @R1	INC R0	INC R1	INC R2	INC R3	INC R4	INC R5	INC R6	INC R7
1	JBC bit, rel	ACALL* addr11	LCALL addr16	RRC A	DEC A	DEC direct	DEC @R0	DEC @R1	DEC R0	DEC R1	DEC R2	DEC R3	DEC R4	DEC R5	DEC R6	DEC R7
2	JB bit, rel	AJMP* addr11	RET	RL A	ADD A, #data	ADD A, direct	ADD A, @R0	ADD A, @R1	ADD A, R0	ADD A, R1	ADD A, R2	ADD A, R3	ADD A, R4	ADD A, R5	ADD A, R6	ADD A, R7
3	JNB bit, rel	ACALL* addr11	RETI	RLC A	ADDC A, #data	ADDC A, direct	ADDC A, @R0	ADDC A, @R1	ADDC C, A, R0	ADDC A, R1	ADDC A, R2	ADDC A, R3	ADDC A, R4	ADDC A, R5	ADDC A, R6	ADDC A, R7
4	JC rel	AJMP* addr11	ORL direct, A	ORL direct, #data	ORL A, #data	ORL A, direct	ORL A, @R0	ORL A, @R1	ORL A, R0	ORL A, R1	ORL A, R2	ORL A, R3	ORL A, R4	ORL A, R5	ORL A, R6	ORL A, R7
5	JNC rel	ACALL* addr11	ANL direct, A	ANL direct, #data	ANL A, #data	ANL A, direct	ANL A, @R0	ANL A, @R1	ANL A, R0	ANL A, R1	ANL A, R2	ANL A, R3	ANL A, R4	ANL A, R5	ANL A, R6	ANL A, R7
6	JZ rel	AJMP* addr11	XRL direct, A	XRL direct, #data	XRL A, #data	XRL A, direct	XRL A, @R0	XRL A, @R1	XRL A, R0	XRL A, R1	XRL A, R2	XRL A, R3	XRL A, R4	XRL A, R5	XRL A, R6	XRL A, R7
7	JNZ rel	ACALL* addr11	ORL C, bit	JMP @A+ DPTR	MOV A, #data	MOV direct, #data	MOV @R0, #data	MOV @R1, #data	MOV R0, #data	MOV R1, #data	MOV R2, #data	MOV R3, #data	MOV R4, #data	MOV R5, #data	MOV R6, #data	MOV R7, #data
8	SJMP rel	AJMP* addr11	ANL C, bit	MOVC A, @A+ PC	DIV A, B	MOV direct, direct	MOV direct, @R0	MOV direct, @R1	MOV direct, R0	MOV direct, R1	MOV direct, R2	MOV direct, R3	MOV direct, R4	MOV direct, R5	MOV direct, R6	MOV direct, R7
9	MOV DPTR, data16	ACALL* addr11	MOV bit, C	MOVC A, @A+ DPTR	SUBB A, #data	SUBB A, direct	SUBB A, @R0	SUBB A, @R1	SUBB A, R0	SUBB A, R1	SUBB A, R2	SUBB A, R3	SUBB A, R4	SUBB A, R5	SUBB A, R6	SUBB A, R7
A	ORL C, /bit	AJMP* addr11	MOV C, bit	INC DPTR	MUL AB	-	MOV @R0, direct	MOV @R1, direct	MOV R0, direct	MOV R1, direct	MOV R2, direct	MOV R3, direct	MOV R4, direct	MOV R5, direct	MOV R6, direct	MOV R7, direct

$\begin{smallmatrix} h \\ \downarrow \\ b \end{smallmatrix}$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B	ANL C, /bit	ACALL* addr11	CPL bit	CPL C	CJNE A, #data, rel	CJNE A, direct, rel	CJNE @R0, #data, rel	CJNE @R1, #data, rel	CJNE R0, #data, rel	CJNE R1, #data, rel	CJNE R2, #data, rel	CJNE R3, #data, rel	CJNE R4, #data, rel	CJNE R5, #data, rel	CJNE R6, #data, rel	CJNE R7, #data, rel
C	PUSH direct	AJMP* addr11	CLR bit	CLR C	SWAP A	XCH A, direct	XCH A, @R0	XCH A, @R1	XCH A, R0	XCH A, R1	XCH A, R2	XCH A, R3	XCH A, R4	XCH A, R5	XCH A, R6	XCH A, R7
D	POP direct	ACALL* addr11	SETB bit	SETB C	DA A	DJNZ direct, rel	XCHD A, @R0	XCHD A, @R1	DJNZ R0, rel	DJNZ R1, rel	DJNZ R2, rel	DJNZ R3, rel	DJNZ R4, rel	DJNZ R5, rel	DJNZ R6, rel	DJNZ R7, rel
E	MOVX A, @DPTR	AJMP* addr11	MOVX A, @R0	MOVX A, @R1	CLR A	MOV A, direct	MOV A, @R0	MOV A, @R1	MOV A, R0	MOV A, R1	MOV A, R2	MOV A, R3	MOV A, R4	MOV A, R5	MOV A, R6	MOV A, R7
F	MOVX @DPTR, A	ACALL* addr11	MOVX @R0, A	MOVX @R1, A	CPL A	MOV direct, A	MOV @R0, A	MOV @R1, A	MOV R0, A	MOV R1, A	MOV R2, A	MOV R3, A	MOV R4, A	MOV R5, A	MOV R6, A	MOV R7, A

Obs: R7-R0 - registrele bancului curent selectat

direct - adresa unui octet din memoria RAM internă

@R0, @R1 - octet din memoria RAM internă adresat indirect prin R0, respectiv R1

#data - constantă de 8 biți inclusă în instrucțiune

#data16 - constantă de 16 biți inclusă în instrucțiune

addr16 - adresă de 16 biți, utilizată de LJMP și LCALL pentru salt/apel în spațiul de 64 Kocteți al memoriei program

addr11 - cei mai puțin semnificativi 11 biți ai adresei de pagină utilizată de AJMP și ACALL pentru salt/apel în pagina de 2Ko curentă (v. §6.5.1).

(*) Cei mai puțin semnificativi 8 biți formează al doilea octet al instrucțiunii, iar următorii 3 biți sunt incluși în codul operației, de aceea instrucțiunile AJMP și ACALL apar fiecare de câte 8 ori în tabel.

rel - octet cu semn în complement față de 2, cu valori cuprinse între -128 și 127, utilizat de instrucțiunile SJMP și de salt condiționat ca offset față de primul octet al instrucțiunii următoare

bit - bit adresat direct din memoria RAM internă sau din zona SFR.

Bibliografie

1. Altman, L., Scrupski, E.S., *Applying Microprocessors*, McGraw-Hill, New York, 1976
2. Blakeslee, R.Th., *Digital Design with Standard MSI & LSI*, 2nd Ed., John Wiley and Sons, Inc., New York, 1979
3. Burileanu, C., *Arhitectura microprocesoarelor*, Ed. Denix, București, 1994
4. Burileanu, C., Niculiu, T., *Limbaje de programare*, Ed. Denix, București, 1994
5. Căpățână, O., Cornea-Hășegan, M., Pușcă M., *Proiectarea cu microprocesoare*, Ed. Dacia, Cluj-Napoca, 1983
6. Căpățână, O., *Proiectarea cu microcalculatoare integrate*, Ed. Dacia, Cluj-Napoca, 1992
7. Coffron, W.J., *Z80 Applications*, Sybex, Berkeley, 1983
8. Coffron, W.J., *Practical Hardware Details for 8080, 8085, Z80, and 6800 Microprocessor Systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981
9. Colin, A., *Programming for Microprocessors*, Newnes-Butterworths, London, 1979
10. Cornea-Hășegan, M., Cornea-Hășegan, D., *Proiectarea sistemelor cu microprocesorul Z80*, Ed. Dacia, Cluj-Napoca, 1988
11. Dancea, I., *Microprocesoare. Arhitectură internă, programare, aplicații*, Ed. Dacia, Cluj-Napoca, 1979
12. Davidoviciu, A., Diatcu, E. ș.a., *Mini- și microcalculatoarele în conducerea proceselor industriale*, Ed. Tehnică, București, 1983
13. Dubois, R., *Famillies 8086-8088 et Z8000 et leur coupleurs*, Eyrolles, Paris, 1985
14. Hall, V.D., *Microprocessors and Interfacing: Programming and Hardware*, 2nd Ed., McGraw-Hill, New York, 1992
15. Hintz, J.K., Tabak, D., *Microcontrollers. Architecture, Implementation, and Programming*, McGraw-Hill Inc., New York, 1992
16. Huțanu, C., *Circuite logice și comenzi secvențiale*, Ed. Junimea, Iași, 1983
17. Huțanu, C., Pănescu, D., "Microsystem with Microprocessor for Digital Control", *Bul. Inst. Politehnic Iași*, fasc.1-2, pp. 92-97, 1981
18. Huțanu, C., Pănescu, D., "Unele aplicații cu microprocesorul CDP1802 COSMAC", *Lucrările I-ului Simpozion Național "Microprocesoare, microcalculatoare, aplicații"*, București, pp.325-330, 1981
19. Huțanu, C., Pănescu, D., Dumbravă, Șt., "Automat programabil cu microprocesor monobit", *Lucrările Sesiunii Științifice jubiliare a Fac. de Electrotehnică*, Iași, 16-17 mai, secțiunea XII, pp. 23-30, 1986
20. Huțanu, C., Pănescu, D., Ganciu, Th., "A Microprocessor System for Frequency Control in Power Systems", *Bul. Inst. Politehnic Iași*, fasc. 1-4, secțiunea IV, pp. 33-38, 1991
21. Huțanu, C., Postolache, M., Pricop, A., Pănescu, D., "Low Cost Automation Distributed Systems", *Proc. on Distributed Control '94 Workshop*, Prague, pp. 69-74, May 19-20, 1994
22. Ionescu, T., *Sisteme și echipamente pentru conducerea proceselor*, E.D.P., București, 1982
23. Kieser, H., Meder, M., *Mikroprozessortechnik*, VEB Verlag Technik, Berlin, 1986
24. Klingman, E.E., *Microprocessor Systems Design*, Prentice-Hall, Englewood Cliffs, New York, 1977

25. Korn, A.G., *Microprocesoare, microcalculatoare, minicalculatoare* (trad. din lb. engleză), Ed. Tehnică, București, 1982
26. Lupu, C., Tepelea, V., Purice, E., *Microprocesoare. Aplicații*, Ed. Militară, București, 1982
27. Lupu, C., Stăncescu, S., *Microprocesoare. Circuite. Proiectare*, Ed. Militară, București, 1986
28. Lupu, C., *Microprocesoare. 2/4/8 biți*, Editura Militară, București, 1995
29. Melusson, P., *Initiation a la microinformatique. Le microprocesseur*, Ed. Techniques et Scientifiques Françaises, Paris, 1980
30. Moraru, Fl., Atodiroaiei, M., *Programarea microcalculatoarelor în sistemul de operare CP/M*, Ed. Științifică și Enciclopedică, București, 1989
31. Mureșan, T., Strugaru, C. ș.a., *Microprocesorul 8080 în aplicații*, Ed. Facla, Timișoara, 1981
32. Patrubany, M., *Totul despre microprocesorul Z80, Vol. 1 și 2*, Ed. Tehnică, București, 1989
33. Pleter, O., Constantinescu, C., *Z80 în SPECTRUM*, Editura Militară, București, 1995
34. Revellin, G., *Microprocesseurs: du 6800 au 6809. Modes d'interfacage*, Dunod, Paris, 1981
35. Sinha, P.K., *Microprocessors for Engineers. Interfacing for Real Time Applications*, Ellis Harwood, Chichester, 1987
36. Spânulescu, I., Spânulescu, S., *Circuite integrate digitale și sisteme cu microprocesoare*, Ed. Victor, București, 1996
37. Sztojanov, I. ș.a., *De la poarta TTL la microprocesoare*, Ed. Tehnică, București, 1987
38. Șerbănați, L.D., *Limbaje de programare și compilatoare*, Ed. Academiei, București, 1987
39. Toacșe, Gh., *Introducere în microprocesoare*, Ed. Științifică și Enciclopedică, București, 1985
40. Tocchi, R.J., Laskowski, L.P., *Microprocesseurs and microordinateurs. Matériel and logiciel*, Eyrolles, Paris, 1981
41. Todorean, G., ș.a., *Transputere și procesoare de semnal*, Ed. Microinformatica, Cluj-Napoca, 1993
42. Zaks, R., Lesea, A., *Techniques d'interface aux microprocesseurs*, Sybex, Paris, 1981
43. * * * *Component Data and Catalog*, Intel Corp., 1977, 1980, 1982
44. * * * *Data Book*, Zilog, Inc., 1982, 1983
45. * * * *Z80® Microprocessor Family Databook*, Zilog, Inc., 1995
46. * * * *Z8® Microcontrollers*, Zilog, Inc., 1994
47. * * * *Intel Pentium Pro Family, Developer's Manual, Vol. 1-3*, Intel Corp., 1996
48. * * * *Pentium® II Processor at 233MHz, 266 MHz, and 300MHz, Data Sheet*, Intel Corp., 1997
49. * * * *Embedded Applications Handbook*, Intel Corp., 1990, 1995
50. * * * *MCS® 51 Microcontroller, User's Manual*, Intel Corp., 1994
51. * * * *MCS® 51 8-Bit Control-Oriented Microcontroller, Data Sheet*, Intel Corp., 1994
52. * * * *Intel MCS Tools Handbook, 3rd ed.*, Intel Corp., 1995
53. * * * *MCS® 151 and MCS® 251 Microcontrollers, Backgrounder*, Intel Corp., 1996
54. * * * *Embedded Microcontrollers*, Intel Corp., 1997
55. * * * *Embedded Applications*, Intel Corp., 1997/1998



Editura ACADEMICA

ISBN 973-97816-3-2